Average-Case Analyse parametrisierter und probabilistischer Algorithmen

Dissertation

zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von Dipl.-Math. Christian Hercher
geboren am 06. 12. 1985 in Jena

Gutachter

1. Prof. Dr. Martin Mundhenk, Friedrich-Schiller-Universität Jena

2. Prof. Dr. Tobias Friedrich, Hasso-Plattner-Institut Potsdam

3. Prof. Dr. Rolf Niedermeier, Technische Universität Berlin

Tag der öffentlichen Verteidigung: 29. 05. 2017

Danksagung

Hiermit möchte ich mich sehr herzlich bei Prof. Tobias Friedrich für die über Jahre währende Unterstützung bei der Entwicklung der Ideen, die in dieser Dissertation vorgelegt werden, bedanken. Auch Prof. Martin Mundhenk gebührt großer Dank, dass er die Betreuung dieses Promotionsvorhabens übernommen hat, als Prof. Friedrich an eine andere Universität wechselte.

Schließlich möchte ich mich auch bei meiner Familie bedanken, die mich nicht nur während der Korrekturphase sondern generell auf meinem Lebensweg immer unterstützt hat.

Inhaltsverzeichnis

Abstract

In both Theoretical Computer Science and practical work it is a disappointing outcome if the considered problem is known to be NP complete. There is almost no hope for the existence of an efficient algorithm solving this problem. However, many approaches have been developed to overcome this barrier:

- The study of parameterized complexity allows in many cases the concentration of the combinatorial explosion of the running time in a given parameter. Then this parameter is hopefully small for the instances to solve.
- The behavior of problems and algorithms not only in the worst case but also in average cases are studied: maybe difficult instances are rare and mostly do not occur in real world situations. Or the data to work with is because of its measurement of physical values slightly perturbed. Then the concept of a smoothed analysis of the algorithm gives an inside: can one achieve expected fast computation for "nearly worst cases", even if in the worst case one only has a much more poorer estimation?
- Also sometimes the use of randomness in the computing process can help to circumvent some obstacles.
- And maybe an approximation is also nearly as good as an optimal solution. So how to find good approximations and how good are they?

All these approaches are well studied on its own, but to the knowledge of the author interactions between them, and the use of multiple approaches together, is a mostly unstudied field of research. In this thesis we want to study a part of these interactions for some test problems.

We show that the reduction rules, given by Gramm et al., for the Clique-Cover problem with high probability not only reduces "yes" instances, but solves them entirely.

We also consider the paradigm of bounded search trees, which is widely used to construct algorithms for fixed parameter tractable problems. There we not only find that the expected running time of a simple bounded search tree algorithm is much lower than the worst case bound for FPT problems Vertex-Cover and d-Hitting-Set. They give for certain sets of parameter values expected FPT running time for the W[1] and W[2] complete problems Clique and Hitting-Set, too.

Furthermore, we study some simple probabilistic generalizations of greedy approximation algorithms. For the Vertex-Cover, Hitting-Set, and the Triangle-Vertex-Deletion problems we find that the probabilistic greedy algorithms we give have a substantially smaller expected approximation ratio than their deterministic worst case equivalents. There is also a trade off between quality and time, meaning that with more time invested one can expect better solutions with smaller approximation ratios.

1 Einleitung

Ein Problem, welches man untersucht, stellt sich als NP-vollständig heraus. Sehr wahrscheinlich gibt es also (da nach Überblick des Autors die gesamte Wissenschaftswelt davon ausgeht, dass $\mathsf{P} \neq \mathsf{NP}$ ist) keinen „schnellen“ Algorithmus, der das Problem löst. Was also tun? Sich mit dieser eher unbefriedigenden Antwort der klassischen (*Worst-Case*)-Komplexitätstheorie abzugeben und die Aufgabe ungelöst zurückzulassen, ist insbesondere in der Praxis oft kein denkbarer Umgang mit der Situation.

Doch es gibt einige Ansätze, wie man sich dennoch diesen „schwierigen“ Problemen nähern kann:

- Man kann versuchen, die Schwierigkeit des Problems (die die Laufzeit explodieren lässt), auf einen (in den relevanten Fällen hoffentlich kleinen) Parameter zu begrenzen. Diese Überlegungen rühren aus dem Feld der *parametrisierten bzw. multivariaten Komplexitätstheorie*.
- Nur weil ein Problem nicht für jede Eingabe schnell gelöst werden kann, heißt dies noch lang nicht, dass dies für alle möglichen Eingaben der Fall sein muss. Vielleicht sind die komplizierten Instanzen ja auch selten, sodass im Durchschnitt dennoch eine effiziente Lösung möglich ist? Derartige Fragen versucht man mit der Betrachtung einer *Average-Case*-Analyse zu beantworten.
- Oder sind die Eingabe-Daten zwar nicht völlig zufällig, aber als Messwerte etwa eventuell etwas verrauscht? Hier kommt die *Smoothed*-Analyse ins Spiel und kann im besten Fall zeigen, dass dadurch im Erwartungswert dennoch schnelle Laufzeiten zu erhalten sind.
- Und wenn der Zufall nicht in den Daten steckt, so kann man trotzdem den Algorithmus dessen Eigenschaften nutzen lassen. Wenn die schwierigen Instanzen deshalb schwierig sind, weil ein deterministischer Algorithmus sich gar nicht immer richtig entscheiden kann, so könnte ein *probabilistischer Algorithmus* dies umgehen, indem er mit genügend hoher Wahrscheinlichkeit richtig rät.
- Oder man gibt sich auch mit einer nicht optimalen Approximationslösung zufrieden. Wie weit diese von der bestmöglichen Lösung abweichen kann, oder – positiv formuliert – welche Garantie man erhält, wie gut die gefundene Näherungslösung ist, untersucht man beim Studium solcher *Approximations-Algorithmen*.

Die genannten Ansätze sind – jeder für sich – gut verstanden und viele Wissenschaftler arbeiten seit längerem in diesen Gebieten. Jedoch sind Verknüpfungen zwischen ihnen teilweise noch kaum untersucht. Fragen wie „Lassen sich Probleme, die auch aus Sicht der parametrisierten Komplexität schwer sind, besser lösen, wenn der Algorithmus nur im Erwartungswert schnell sein muss?“ oder „Kann ein probabilistischer Algorithmus bessere Näherungslösungen produzieren, als man bei deterministischem Vorgehen erhält?“ erscheinen noch keine große Beachtung erhalten zu haben.

Der Autor geht jedoch davon aus, dass die verschiedenen, genannten Ansätze sehr wohl deutlich voneinander profitieren können. Selbst da, wo die Methoden eines Bereichs scheitern, können durch Anwenden der Techniken der verschiedenen Ansätze teils deutlich bessere Ergebnisse erzielt werden.

Diese Dissertation soll das Zusammenwirken der genannten Techniken an einigen ausgesuchten Beispielen demonstrieren. In Kapitel 2 werden die für diese Arbeit grundlegenden Bereiche und wichtige Informationen daraus noch einmal kurz vorgestellt.

Es folgt in Kapitel 3 die Diskussion, wie gut gewisse Reduktionsregeln, die ein Problem auf seinen Problemkern reduzieren, für das Clique-Cover-Problem sind: Bekannt war, dass sie die Eingabe im Worst-Case auf einen Problemkern aus höchstens 2^k Knoten reduzieren, wenn eine Überdeckung des Graphen durch k Cliquen existiert. Wir können zeigen, dass unter Annahme einer sinnvollen Eingabeverteilung diese Reduktionsregeln das Problem nicht nur deutlich vereinfachen, sondern sogar schon mit hoher Wahrscheinlichkeit selbst lösen.

Im Kapitel 4 wird das Paradigma der beschränkten Suchbäume bei der Anwendung auf die in der parametrisierten Komplexitätsklasse FPT liegenden Probleme Vertex-Cover und d-Hitting-Set, das W[1]-vollständige Problem Clique sowie das W[2]-vollständige Problem Hitting-Set betrachtet: Wie verhält es sich, wenn die Eingaben gemäß eines recht allgemeinen Zufalls-Graph- (bzw. Zufalls-Mengen-) Modells verteilt sind? Dabei konnten wir für die beiden schon im Worst-Case zur Klasse der FPT-Probleme gehörenden Fälle deutlich schnellere Laufzeiten im Erwartungsfall zeigen, während wir auch für die beiden anderen Probleme bei geeigneter Parameter-Wahl eine erwartete FPT-Laufzeit zeigen konnten. Durch die Wahl des allgemeinen Zufalls-Graph- (bzw. -Mengen-) Modells ließen sich diese Erkenntnisse auch jeweils als Aussagen im Sinne der Smoothed-Komplexität formulieren, sodass diese Probleme nicht nur im Average-Case, sondern auch bei nur geringeren Störungen noch effektiv lösbar sind.

Abschließend betrachten wir in Kapitel 5 für das Vertex-Cover-Problem, das d-Hitting-Set-Problem, wobei d die Größe der zu treffenden Mengen angibt, und das Triangle-Vertex-Deletion-Problem jeweils den (deterministischen) Greedy-Approximations-Algorithmus und verallgemeinern diesen, indem die Auswahl des zu ziehenden Elements nicht mehr gierig (engl. greedy) als das nach einer bestimmten Bewertung beste erfolgt, sondern probabilistisch mit einer Wahrscheinlichkeitsverteilung, die von der Bewertung der verschiedenen Elemente abhängt. Auch hier konnte für alle drei Probleme eine deutliche Verbesserung gegenüber dem deterministischen Greedy-Algorithmus gezeigt werden: Während dieser im Worst-Case nur eine in der Knoten- bzw. Mengen-Anzahl n logarithmische Approximation zulässt, konnte für die jeweils betrachtete Wahrscheinlichkeitsverteilung bewiesen werden, dass im Erwartungswert ein konstanter Approximationsfaktor (der vom Problem abhängt) erreicht wird. Dabei lässt sich in einer Kosten-Nutzen-Abwägung zwischen erwarteter Laufzeit und Approximationsgüte diese Konstante auch beliebig weit nahe 1 verschieben.

2 Grundlagen

Dieses Kapitel soll einen kurzen Überblick über die Gebiete, in denen sich diese Dissertation bewegt, geben. Es ist dabei nicht als Ersatz eines Lehrbuchs gedacht. Entsprechende, ausführlichere Literatur findet man jeweils in den einzelnen Abschnitten angegeben.

2.1 Worst-Case-Komplexitäts-Theorie

Die Theoretische Informatik beschäftigt sich neben der Frage, was überhaupt berechenbar ist, auch intensiv mit Überlegungen dazu, wenn etwas berechenbar ist, wie schnell man das jeweilige Problem denn auch lösen kann. Dabei stellt sich heraus, dass die Probleme in eine ganze Hierarchie von Komplexitätsklassen eingeordnet werden können. Deren Studium widmet sich die Komplexitäts-Theorie.

Einführungen in die Theoretische Informatik und insbesondere die Komplexitätstheorie findet man unter anderem in Blum [15], Schöning [92], Wechsung [99] sowie auch z.B. in [19, 40, 61, 62]. Wir wollen hier im Folgenden nur einen kurzen Überblick geben.

Eine wichtige – wenn nicht gar die erste und zentrale – Frage, die sich mit dem Aufkommen von programmierbaren Rechenmaschinen ab dem vierten Jahrzehnt des zwanzigsten Jahrhunderts verstärkt stellte, ist, welche Probleme sich mit diesen – und generell – berechnen lassen, und was auch der beste Computer aus inhärenten, theoretischen Gründen nicht wird lösen können. Doch um diese Frage beantworten zu können, benötigt es erst einen gewissen Formalismus: Was ist überhaupt ein Problem? Und was heißt es, dass ein solches (nicht) berechenbar ist?

Grundlegend, quasi als atomarer Baustein des Theorie-Gebäudes, ist dabei der Begriff der *Sprache*. Dabei ist keine natürliche oder künstliche Sprache zur Verständigung gemeint, sondern ein deutlich abstrakterer Begriff:

Definition 2.1.1. Sei Σ eine endliche, nichtleere Menge. Dann nennen wir Σ ein *Alphabet* und seine Elemente *Buchstaben*. Endliche Folgen von Buchstaben (insbesondere auch die aus 0 Buchstaben) heißen *Wörter*. Die Menge aller Wörter wird mit Σ^* bezeichnet. Eine beliebige Teilmenge $L \subset \Sigma^*$ heißt *Sprache*.

Eine Sprache ist also eine bestimmte, wohldefinierte Menge von endlich langen Zeichenketten, die ihrerseits aus nur endlich vielen verschiedenen Zeichen bestehen. Beispielsweise könnte man als Alphabet die Menge der Ziffern 0 bis 9 wählen und die Sprache PRIMES der Dezimaldarstellung der Primzahlen. Aber es sind auch ganz andere Sprachen vorstellbar. So lassen sich auch z. B. Graphen in entsprechende Zeichenketten codieren und man kann die Sprache HAMILTON derjenigen Graphen, die einen Hamilton-Weg besitzen, definieren.

Probleme können nun als *Wortproblem* definiert werden: Ist ein gegebenes Wort x Element der Sprache L?

Um solche Fragen zu berechnen, bzw. erst einmal die Frage zu klären, ob sie überhaupt berechenbar ist, benötigt es ein Berechenbarkeitsmodell. Dabei stehen viele zur Auswahl: Von verschiedenen Programmiersprachen über Random Access Machines, μ-rekursiven Funktionen und dem λ-Kalkül bis hin zu Conways Game of Life: Sie alle sind äquivalent, d. h., können genau die gleichen Dinge berechnen. Nach der Turing-Church-These ist das deshalb der Fall, da sie alle mit dem (nicht klar definiertem) intuitivem Berechenbarkeitsbegriff übereinstimmen (und nicht nur – wie z. B. die Klasse der primitiv-rekursiven Funktionen – eine echte Teilmenge davon darstellen).

Als das Berechenbarkeitsmodell schlechthin hat sich aber das von einem der Urväter der Disziplin der Theoretischen Informatik – Alan Turing (1912–1954) – stammende und 1936 vorgestellte Modell einer hypothetischen Rechenmaschine etabliert; die nach ihm benannte Turing-Maschine:

Definition 2.1.2. Eine deterministische / nicht-deterministische Turingmaschine ist gegeben durch ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$. Dabei ist

- Z die endliche *Zustandsmenge*,
- Σ das *Eingabealphabet*,
- $\Gamma \supset \Sigma$ das *Arbeitsalphabet*,
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ im deterministischen Fall bzw. $\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$ im nicht-deterministischen Fall die *Überführungsfunktion*,
- $z_0 \in Z$ der *Startzutand*,
- $\square \in \Gamma \setminus \Sigma$ das *Leerzeichen* und
- $E \subset Z$ die Menge der *Endzustände*.

Definition 2.1.2 ist aus [92] übernommen.

Um für diese abstrakte Definition eine Anschauung zu entwickeln, was sie konkret bedeutet und wie eine Turing-Maschine arbeitet, stellt man sich ein beidseitig unendlich langes Band vor, welches in einzelne, aufeinander folgende Zellen unterteilt ist. In jeder dieser Zellen steht ein einzelnes Zeichen des Bandalphabets Γ, wobei fast alle Zellen (d. h. alle, bis auf endlich viele Ausnahmen) durch das Leerzeichen $\square$ beschriftet sind. (Es gibt also nur einen endlichen Bereich, in dem andere Zeichen stehen können. Insbesondere wird dort die Eingabe durch die Teilmenge des Eingabealphabets Σ notiert.)

Neben diesem Speicher besitzt die Turingmaschine auch einen Lese-Schreib-Kopf, der über einer entsprechenden Zelle steht, deren Inhalt auslesen und einen neuen hineinschreiben kann. Pro Takt kann dieser die beiden Aktionen (Lesen und Schreiben) ausführen und sich dann abschließend auf eine der beiden Nachbarzellen bewegen.

Diese Bewegung wird durch die Richtung angegeben: L für die linke Nachbarzelle, R für die rechte und N für keine Bewegung („Nicht bewegen“).

Schlussendlich fehlt noch eine Steuereinheit, die angibt, wie sich die Turing-Maschine verhält. Zu Beginn befindet sich die Maschine im Start-Zustand z_0. Abhängig vom aktuellen Zustand aus der Zustandsmenge Z und dem in diesem Takt gelesenen Zeichen werden nun drei Aktionen ausgeführt:

- Es wird ein neues Zeichen in die aktuelle Zelle geschrieben,
- der Lese-Schreib-Kopf wird nach links oder rechts auf die Nachbarzelle, oder gar nicht bewegt und
- der Zustand der Turing-Maschine wird aktualisiert.

Welches Zeichen genau geschrieben, in welche Richtung der Lese-Schreib-Kopf bewegt und welcher neue Zustand angenommen wird, regelt die Überführungsfunktion δ. (Man beachte die verschiedenen Definitionen von δ deterministischen bzw. nicht-deterministischen Fall.) Für eine deterministische Turing-Maschine ist für jedes Paar aus aktuellem Zustand und gelesenem Zeichen genau eine Möglichkeit vorgegeben, d. h. eindeutig bestimmt, welches Zeichen geschrieben, wie der Lese-Schreib-Kopf bewegt und in welchen neuen Zustand die Turing-Maschine für den nächsten Takt gehen soll. Für eine nicht-deterministische Turing-Maschine stehen hier prinzipiell auch mehrere solche Möglichkeiten zur Verfügung. Eine nicht-deterministische Berechnung folgt dann einem der dadurch möglichen Berechnungspfade. Nach Abschluss der Ausführung dieser drei gerade genannten Aktionen ist der aktuelle Takt abgeschlossen und die Turing-Maschine berechnet nun den nächsten Takt (d. h., liest das aktuelle Zeichen und bestimmt mittels des aktuellen Zustands und der Überführungsfunktion die nächsten Aktionen usw.).

Diese taktweise Abarbeitung wird solange wiederholt, bis die Turing-Maschine einen Zustand, der Teil der Endzustandsmenge E ist, erreicht. Damit hält die Turing-Maschine und die Berechnung ist beendet.

Zerlegt man darüber hinaus die Endzustandsmenge disjunkt in die zwei Teilmengen $E = E_{an} \dot\cup E_{ab}$, so kann man zusätzlich auch durch Betrachtung des erreichten Endzustands codieren, ob eine bestimmte Eingabe „angenommen“ oder „abgelehnt“ wurde.

Definition 2.1.3. Sei M eine (nicht-)deterministische Turing-Maschine. Dann heißt die Menge $L(M)$ der Eingaben, für die M in einem akzeptierenden Endzustand hält, die von M akzeptierte Sprache.

Nun hat man für die Frage, wie man die Berechenbarkeit eines Problems formal definiert, alles zusammen: Ein Problem lässt sich als Wortproblem einer Sprache L auffassen. Und eine Turing-Maschine M löst dieses genau dann, wenn L die von M akzeptierte Sprache ist, aber M (dann in ablehnenden Endzuständen) auch für alle Eingaben $x \notin L$ hält.

Nicht alle Probleme sind berechenbar. So ist die Frage, ob eine bestimmte Turing-Maschine bei einer bestimmten Eingabe hält – das HALTEPROBLEM – ein solches Problem, wofür es keine Turing-Maschine geben kann, die es berechnet. (Der Beweis geschieht über ein Diagonalisierungs-Argument.)

Doch bei den Problemen, die berechenbar sind, interessiert eine weitere Frage: Wie effizient ist die Berechnung möglich? Besonders die Frage, ob eine Berechnung in polynomieller Zeit (in Abhängigkeit von der Länge $|x|$ der Eingabe x) möglich ist, ist von großem Interesse. Sie entscheidet im weiteren Sinne darüber, ob das Problem „praktisch lösbar“ oder „praktisch unlösbar“ ist, da im zweiten Fall die Laufzeiten mit zunehmender Eingabelänge förmlich explodieren.

Definition 2.1.4. Die Sprache L liegt in der Komplexitätsklasse P genau dann, wenn es ein Polynom p und eine (deterministische) Turing-Maschine M gibt, sodass $L(M) = L$ ist und für jede Eingabe x eine Laufzeit (Anzahl der Takte bis zum Halten der Turing-Maschine) von höchstens $p(|x|)$ besitzt.

Analog besteht die Komplexitätsklasse NP aus genau den Problemen, für die eine nicht-deterministische Turing-Maschine M mit $L(M) = L$ und ein Polynom p existiert, sodass es für jede akzeptierte Eingabe x einen möglichen akzeptierenden Berechnungspfad gibt, dessen Länge durch $p(|x|)$ beschränkt ist.

Man bemerke zuerst, dass die Einschränkung auf akzeptierende Eingaben im Fall der deterministischen Turing-Maschine keinen Unterschied macht: Auch die nicht in der Sprache L liegenden Wörter x werden dabei als solche erkannt: Nach spätestens $p(|x|)$ Schritten hätte ja sonst die Eingabe akzeptiert werden müssen. Ist dies bis dahin nicht geschehen, dann kann die Berechnung abgebrochen und in einen ablehnenden Endzustand gegangen werden. (Dies ist die informelle Beschreibung der Arbeitsweise einer Turing-Maschine M', die höchstens einen konstanten Faktor längere, also noch immer polynomielle Laufzeit besitzt, aber auch auf den $x \notin L$ hält, obwohl dies für M nach obiger Definition nicht gefordert war.)

Darüber hinaus führe man sich vor Augen, dass im nicht-deterministischen Fall die Definition jedoch sehr großzügig ist: Für Eingaben x, die nicht Element der akzeptierten Sprache sind, ist gar keine Vorgabe gemacht, d. h., die Turing-Maschine muss nicht einmal halten. (Zwar kann man wieder, analog der Überlegung im deterministischen Fall eine Turingmaschine M' konstruieren, die die Berechnung nach genügend langer Zeit abbricht. Aber diese hat nun exponentielle Laufzeit...) Und auch für Elemente der betrachteten Sprache muss nur ein solcher Berechnungs-Pfad, der polynomielle Länge hat, existieren. Wie man diesen (aus der Vielzahl der Möglichkeiten) findet, oder ob andere länger sind, wird dabei nicht betrachtet.

Damit erhält man auch eine äquivalente Charakterisierung der Komplexitätsklasse NP:

Satz 2.1.5. *Die Komplexitätsklasse* NP *besteht aus genau den Sprachen L, für die es für jedes $x \in L$ ein polynomiell beschränktes Zertifikat für dessen Mitgliedschaft in L gibt.*

Ein *Zertifikat* ist dabei ein Nachweis, der genau dem in Definition 2.1.4 geforderten akzeptierenden Berechnungspfad entspricht. Viele praktische Probleme lassen sich so der Komplexitätsklasse NP zuordnen: Wenn man einen Lösungskandidaten hat (etwa einen Weg in einem Graphen, der einen Hamilton-Weg darstellt), dann kann man ihn auch leicht verifizieren. Die Frage ist nur, wie man diese Kandidaten findet. Gelingt das effizient, sodass das Problem auch in P liegt?

Um die Beziehungen zwischen solchen Problemen untersuchen zu können, führt man eine Relation ein, die anschaulich als „ist höchstens so schwer wie“ interpretiert werden kann; die *Polynomialzeit-beschränkte many-one-Reduktion*:

Definition 2.1.6. Es seien $A, B \subset \Sigma^*$ zwei Sprachen. Dann heißt A in Polynomialzeit many-one-reduzierbar auf B ($A \leq_p B$), wenn es eine Polynomialzeit Turing-berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ mit $a \in A \Leftrightarrow f(a) \in B$ gibt. Eine Funktion f heißt dabei in Polynomialzeit Turing-berechenbar, wenn es eine deterministische Turing-Maschine M und ein Polynom p gibt, sodass M für jede Eingabe $x \in \Sigma^*$ nach höchstens $p(|x|)$ Schritten auf der Ausgabe $f(x)$ hält.

Um die Frage, ob eine Eingabe a Element von A ist, zu beantworten, genügt es also zu überprüfen, ob $f(a) \in B$ gilt. Kann man dies deterministisch in polynomieller Zeit ermitteln, so ist dies demnach auch für die Ausgangsfrage der Fall. Das Problem A ist damit „höchstens so schwer“ wie das Problem B.

Definition 2.1.7. Sei S eine Sprache. Gilt für jede Sprache $L \in \mathsf{NP}$, dass $L \leq_p S$, so heißt S NP-schwer. Ist zusätzlich noch S selbst Element von NP, so heißt es sogar NP-vollständig.

Die NP-Vollständigkeit stellt einen sehr zentralen Begriff dar: Ist ein Problem NP-vollständig, dann bedeutet dies, dass, wenn man jenes eine Problem effizient (d. h. deterministisch in Polynomialzeit) lösen könnte, dann ginge das mit allen Problemen aus NP.

Ein erstes solches NP-vollständiges Problem konnte 1971 gefunden werden, siehe [29, 70]:

Satz 2.1.8 (Satz von Cook und Levin)**.** *Das* SAT*-Problem, ob für eine boolesche Formel F eine Belegung der Variablen existiert, die F erfüllt, ist* NP*-vollständig.*

Damit war ein Anfang gemacht. Doch bald folgte für viele weitere Probleme die Einordnung, dass sie NP-vollständig sind, siehe z. B. Karps berühmte Liste von 21 NP-vollständigen Problemen in [65]. Seitdem wurde diese Eigenschaft für eine große Zahl weiterer Probleme gezeigt.

Doch für keines dieser Probleme konnte bisher ein deterministischer Polynomialzeit-Algorithmus gefunden werden. Gäbe es einen solchen, so würde es nach Definition der NP-Vollständigkeit zum Kollaps der Komplexitätsklassen und damit der Identität $\mathsf{P} = \mathsf{NP}$ führen. Da trotz intensiver Suche kein solcher Algorithmus gefunden wird, ergibt sich die berühmte

Vermutung 2.1.9. Es ist $\mathsf{P} \neq \mathsf{NP}$, also $\mathsf{P} \subsetneq \mathsf{NP}$.

Diese Frage, die schon Cook [29] 1971 formulierte, ist von solcher Bedeutung, dass sie auch vom Clay-Math-Institute im Jahr 2000 zu einem der sieben Millennium-Probleme erhoben wurde, auf deren Lösungen je eine Million US-Dollar Preisgeld ausgeschrieben ist. Dennoch ist diese Frage weiterhin unbeantwortet. Ohne einen wesentlichen Durchbruch ist sie wohl auch nicht zu klären. Damit verbleiben viele Aussagen der Komplexitätstheorie in der Abhängigkeit, dass sie nur unter der (von allen als wahr angenommenen aber eben noch unbewiesenen) Vermutung $\mathsf{P} \neq \mathsf{NP}$ abhängen.

Darüber hinaus gibt es neben P und NP einen ganzen „Zoo“ weiterer Komplexitätsklassen, deren Beziehungen untereinander von großem Interesse sind.

2.2 Parametrisierte Komplexitäts-Theorie

Was macht man nun, wenn das Problem, das man lösen soll, sich als NP-vollständig herausstellt? Eine Alternative bietet die parametrisierte bzw. multivariate Komplexitätstheorie. Hier ist der Ansatz, dass die Beschreibung des Laufzeitverhaltens in Abhängigkeit allein von der Länge der Eingabe nicht ausreicht, um die Schwierigkeit des Problems genügend genau zu beschreiben. Möglicherweise gibt es ja weitere Parameter (bei Graphen etwa der Maximalgrad), die maßgeblichen Einfluss auf die Laufzeit von Algorithmen nehmen können. Im für den Anwender günstigen Fall lässt sich die gesamte Schwierigkeit des Problems auf einen Parameter konzentrieren, welcher aber in den praktischen Anwendungsfällen eher klein ist, sodass sich dennoch eine vertretbare Laufzeit für diese Eingaben ergibt.

Gute Einführungen in das Gebiet der parametrisierten und multivariaten Komplexitätstheorie finden sich unter anderem in Downey und Fellows [37], Niedermeier [80] sowie [42].

Im Bereich der parametrisierten Komplexität sind parametrisierte Sprachen L Gegenstand der Betrachtung, d. h. $L \subset \Sigma^* \times \mathbb{N}$. Dabei stellt bei einem Paar (x, k), bei welchem die Frage beantwortet werden soll, ob es in der parametrisierten Sprache L liegt, der Term x die „normale“ Eingabe (also z. B. einen Graph) und der Wert k den Parameter (z. B. die Größe der gesuchten Lösungsmenge) dar.

Der zentrale Begriff in der Welt der parametrisierten Komplexität ist dabei der der *fixed parameter tractability*, FPT:

Definition 2.2.1. Eine Sprache L heißt für den Parameter k fixed parameter tractable und gehört der entsprechenden Komplexitätsklasse FPT an, wenn es einen (deterministischen) Algorithmus A, eine Konstante c und eine berechenbare Funktion f gibt, sodass für alle x und k der Algorithmus A bei Eingabe (x, k) eine Laufzeit von höchstens $f(k) \cdot |x|^c$ besitzt und $(x, k) \in L$ genau dann gilt, wenn der Algorithmus A diese Eingabe akzeptiert, also $L(A) = L$ ist.

Man lässt dabei also zu, dass die nur vom Parameter abhängige Komponente der Laufzeit möglicherweise beliebig schnell wächst, während allerdings bei gleichbleibendem Parameter die Laufzeit des Algorithmus nur polynomiell in der Eingabelänge $|x|$ wachsen darf.

Für viele NP-vollständige Probleme sind Parametrisierungen bekannt, sodass sie in der Komplexitätsklasse FPT liegen, etwa Vertex-Cover [26], Clique-Cover [54], 3-Hitting-Set [81], Planar-Dominating-Set [4] oder auch verschiedene String-Probleme [22]. Meist ist hierbei der Parameter die Größe der gesuchten Lösungsmenge. Es sind aber auch andere Parameter denkbar. So gibt etwa Courcelles Theorem [30] die Möglichkeit, jedes Problem, dass sich mithilfe der monadischen Logik zweiter Ordnung formulieren lässt, in FPT-Zeit mit dem Parameter der Baumweite des Eingabegraphen zu lösen.

Jedoch gibt es auch viele solche NP-vollständige Probleme, bei denen sich bisher kein solcher FPT-Algorithmus finden ließ. Dies betrifft etwa Clique [35], Hitting-Set [36] und Dominating-Set [34].

Wie in der klassischen Komplexitätstheorie kann man auch hier neben FPT weitere Komplexitätsklassen definieren. Dies kann etwa durch die Betrachtung der Probleme, die durch bestimmte Schaltkreis-Logiken darstellbar sind, geschehen. So ergibt sich oberhalb von FPT eine gesamte Hierarchie von Komplexitätsklassen, die W-Hierarchie genannt wird:

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots$$

Um Beziehungen zwischen diesen Klassen parametrisierter Probleme untersuchen zu können, wird wieder eine der Polynomialzeit-Reduktion ähnliche Reduktion, die diesmal die Lösbarkeit in FPT-Zeit erhalten soll, betrachtet:

Definition 2.2.2. Es seien $A, B \subset \Sigma^* \times \mathbb{N}$ zwei parametrisierte Sprachen. Dann heißt A in FPT-Zeit many-one-reduzierbar auf B ($A \leq_{\mathsf{FPT}} B$), wenn es eine in FPT-Zeit berechenbare Funktion $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times N$ mit $(a,k) \in A \Leftrightarrow f((a,k)) \in B$ gibt. Eine Funktion f heißt dabei in FPT-Zeit berechenbar, wenn es eine Turing-berechenbare Funktion $g : \mathbb{N} \to \mathbb{N}$ und eine Konstante c gibt, sodass für alle Paare $(x,k) \in \Sigma^* \times \mathbb{N}$ die Berechnung von $f((x,k))$ in einer Zeit von höchstens $g(k) \cdot |x|^c$ erfolgen kann.

Damit erhält man auch wieder analog einen entsprechenden Vollständigkeits-Begriff und es stellt sich heraus, dass das Clique-Problem W[1]-vollständig und sowohl das Hitting-Set- wie auch das Dominating-Set-Problem W[2]-vollständig sind.

Ähnlich der P vs. NP - Frage ist auch hier ungeklärt, ob in der W-Hierarchie alle Teilmengenbeziehungen echt sind, oder zwei (oder mehr) Klassen zusammenfallen und in Wahrheit identisch sind. Man geht dabei analog der Vermutung 2.1.9 aber davon aus, dass diese Hierarchie echt ist, also $\mathsf{FPT} \subsetneq \mathsf{W}[1] \subsetneq \mathsf{W}[2] \subsetneq \cdots$ gilt.

2.3 Average-Case-Analyse

Stellen sich Probleme als NP-schwer heraus, so ist das eine Worst-Case-Aussage: Unter der Annahme, dass $\mathsf{P} \neq \mathsf{NP}$ ist, gibt es dann also für jeden Algorithmus eine Folge von Instanzen, deren Bearbeitung nicht in polynomieller Zeit erfolgen kann. Dies macht aber noch keine Aussage darüber, wie häufig diese Instanzen sind. Sind sie z. B. sehr selten, so kann ein Algorithmus dennoch „in der Praxis“ sehr schnell sein, da er eben zumeist nicht auf diese schweren Instanzen angewendet werden muss, sondern eben auf diese nur selten trifft. Umgekehrt möchte man für kryptographische Anwendungen, dass die Ermittlung des Klartexts aus dem Chiffre-Text ohne den verwendeten Schlüssel zu kennen, nicht nur im Worst-Case sondern im „Normalfall“, dem Average-Case schwer, d. h. nicht in polynomieller Zeit lösbar, sein sollte.

Es gibt also gute Gründe, über die Betrachtung der Worst-Case-Komplexität hinaus zuschauen. Gute Einführungen zur Average-Case-Komplexität findet man etwa in [10, 71, 98] sowie eine konkrete Beispiel-Situation für das Sortierverfahren Quick-Sort bei Knuth [67].

Um etwas über eine „durchschnittliche“ Instanz aussagen zu können, benötigt es zuerst eine Angabe über die Verteilung der Eingabe-Instanzen. Dabei sei $\mu = (\mu_n)$ eine Familie von Wahrscheinlichkeitsverteilungen μ_n der Probleminstanzen x mit $|x| = n$. Nun sind Aussagen über das Verhalten von Algorithmen im μ-Average-Case möglich. Um Sprache und Verteilung gleich als zusammengehörige Einheit betrachten zu können, bezeichnet man das Paar (L, μ) einer Sprache und einer entsprechenden Familie von Eingabeverteilungen als *zufallsverteiltes Problem* (engl. distributional problem bzw. randomized problem).

Dabei möchte man nun als Erstes formalisieren, was es heißt, dass ein solches zufallsverteiltes Problem „im Durchschnitt leicht“ ist. Intuitiv sollte dies genau dann gegeben sein, wenn ein Algorithmus mit (bezüglich μ) erwarteter polynomieller Laufzeit für dieses existiert. Um eine maschinen-unabhängige Definition geben zu können, wird die Bedingung leicht anders, aber mit gleichem Inhalt formuliert:

Definition 2.3.1. Die Komplexitätsklasse $\mathsf{avg}-\mathsf{P}$ besteht aus genau den zufallsverteilten Problemen (L, μ), für die ein Algorithmus A, der L akzeptiert, und eine Konstante $0 < \varepsilon$ existieren, sodass für die Laufzeitfunktion $t_A : \Sigma^* \to \mathbb{N}$ von A und alle $1 \leq n \in \mathbb{N}$ die Beziehung

$$\sum_{|x|=n} \mu_n(x) \cdot t_A(x)^\varepsilon \cdot |x|^{-1} < \infty$$

gilt.

Insbesondere fällt dabei auf, dass die Laufzeit nicht für alle Eingaben polynomiell beschränkt sein muss, die Wahrscheinlichkeit für solche Elemente dann aber subpolynomiell (d. h. kleiner als jedes n^c) abnehmen muss, wenn das gesamte Problem noch in $\mathsf{avg}-\mathsf{P}$ liegen soll.

Wir bemerken auch, dass die in späteren Kapiteln dieser Arbeit verwendete Komplexitätsklasse $\mathsf{avg}-\mathsf{FPT}$ parametrisierter zufallsverteilter Probleme sich analog definieren lässt. Dabei teile man in der Summe nicht nur durch $|x|$, sondern auch durch die Funktion $f(k)$, die die Laufzeit-Komponente des Parameters k beschreibt.

Die Komplexitätsklasse $\mathsf{avg}-\mathsf{P}$ stellt hierbei das Average-Case-Äquivalent zur Klasse P der Worst-Case-Komplexitätstheorie dar. Deshalb liegt es nahe, auch ein entsprechendes Äquivalent für die Klasse NP zu finden, was mit $\mathsf{dist}-\mathsf{NP}$ gelingt:

Definition 2.3.2. Die Komplexitätsklasse $\mathsf{dist}-\mathsf{NP}$ besteht aus genau den zufallsverteilten Problemen (L, μ), für die $L \in \mathsf{NP}$ und μ P-berechenbar ist. Dabei heißt eine Familie $\mu = (\mu_n)$ von Wahrscheinlichkeitsverteilungen P-berechenbar, wenn ein Polynom p existiert, sodass für alle $n \in \mathbb{N}$ und $x \in \Sigma^n$ die Verteilungsfunktion

$$\mu_n^*(x) := \sum_{y \in \Sigma^n, y \leq x} \mu(y)$$

in einer durch $p(|x|) = p(n)$ beschränkten Zeit berechenbar ist. Die Wörter seien hierbei lexikographisch geordnet.

Damit besteht also $\mathsf{dist}-\mathsf{NP}$ aus den Problemen in NP, verbunden mit Wahrscheinlichkeitsverteilungen, die in polynomieller Zeit berechenbar sind.

Auch erhalten wir analog wieder einen entsprechenden Reduktions- und damit dann auch Vollständigkeitsbegriff:

Definition 2.3.3. Es seien (A, μ) und (B, ν) zwei zufallsverteilte Probleme. Dann heißt (A, μ) $\mathsf{avg}-\mathsf{P}$-reduzierbar auf (B, ν) (geschrieben als $(A, \mu) \leq_{\mathsf{avg}-\mathsf{P}} (B, \nu)$), wenn eine in Polynomialzeit berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ sowie Polynome p und m existieren, sodass einerseits für alle $x \in \Sigma^*$ gilt, dass $x \in A \Leftrightarrow f(x) \in B$ ist und andererseits für jedes $x \in \Sigma^*$ sowohl $|f(x)| \leq m(|x|)$ als auch die Beziehung

$$\sum_{x : f(x) = y} \mu_{|x|}(x) \leq p(|x|)\nu_{|y|}(y)$$

gilt.

Der zweite Teil der Definition sichert, dass die Wahrscheinlichkeit, eine „schwere“ Instanz zu ziehen, sich bei der Reduktion nur um höchstens einen polynomiellen

Faktor ändert. War diese Wahrscheinlichkeit zuvor subpolynomiell, ist sie es danach auch weiterhin und erhält so die Eigenschaft, im Erwartungswert noch eine polynomielle Laufzeit zu besitzen.

Auch für diesen Reduktionsbegriff wurden dist−NP-vollständige Probleme gefunden, als ein bedeutendes Beispiel das beschränkte Halteproblem (mit der uniformen Verteilung), welches danach fragt, ob eine gegebene Turing-Maschine in $\leq k$ Schritten hält, [57]. Es lässt sich eine gesamte Komplexitätstheorie für Average-Case-Betrachtungen aufbauen. So lässt sich etwa zeigen, dass aus avg−P = dist−NP der Kollaps der beiden Worst-Case-Klassen EXP = NEXP, die eine deterministische bzw. nicht-deterministische exponentielle Laufzeit-Beschränkung besitzen, folgt, [9]. Auch zeigt sich, dass für jedes NP-schwere Problem eine Verteilung existiert, sodass das entsprechende zufallsverteilte Problem dist−NP-schwer ist, [72]. Damit lässt sich so allerdings wenig über „natürliche“ Verteilungen auf den bekannten NP-vollständigen Problemen aussagen. Aber für einzelne dieser gibt es zumindest positive Ergebnisse, wie etwa ein erwarteter $\mathcal{O}(1)$-Algorithmus für die 3-Färbbarkeit von Graphen, [100].

Auf das vielleicht prominenteste Beispiel, Quick Sort, soll abschließend auch noch einmal kurz hingewiesen werden: Hier ist bekannt, dass die naive Implementierung des Verfahrens beim Sortieren von n Elementen eine Worst-Case-Laufzeit von $\mathcal{O}(n^2)$ besitzt. Bei einer Average-Case-Betrachtung (bei der alle Permutationen der Eingabeelemente als gleichwahrscheinlich angesehen werden) stellt sich jedoch heraus, dass die für den Algorithmus „schwierigen“ Instanzen recht selten sind, und man eine erwartete Laufzeit von $\mathcal{O}(n \log n)$ erhält.

2.4 Smoothed-Analyse

Häufig ist für eine Betrachtung der Schwierigkeit eines Problems die klassische Worst-Case-Betrachtung für die Realität zu pessimistisch. (Wie oft tritt ein solcher „schlechter“ Fall überhaupt auf? Oft sind dabei die Worst-Case-Instanzen sehr künstlich und konstruiert . . .) Andererseits erscheint eine nur auf den durchschnittlichen Fall ausgelegte Average-Case-Analyse wiederum zu sehr alle Schwierigkeiten herauszumitteln. Interessant ist daher auch, was zwischen diesen beiden Extremen (Worst- bzw. Average-Case) geschieht: Wie verhält sich der betrachtete Algorithmus, wenn man nur eine kleine Variation der Eingabedaten zulässt?

Dieser Frage widmet sich die Untersuchung der Smoothed-Komplexität. Eine gute Einführung dazu findet man in Spielman und Teng [94, 95].

Wir geben hier eine Version der Smoothed-Komplexität für Graph-Probleme im Kontext unseres „Allgemeinen Zufallsgraph-Modells“, siehe Definition 2.7.4 in Abschnitt 2.7.4, an.

Definition 2.4.1 (Smoothed-Komplexität)**.** Seien $n \in \mathbb{N}$, $\mathcal{G}_n$ die Menge der Graphen mit n Knoten und p eine Wahrscheinlichkeitsfunktion $\mathbb{N} \to \left[0; \frac{1}{2}\right]$. Weiterhin sei G ein fester Graph mit n Knoten und $RG(G,p)$ der Wahrscheinlichkeitsraum über $\mathcal{G}_n$, in welchem jede potentielle Kante in G unabhängig mit Wahrscheinlichkeit p gestört wurde, das heißt, war eine potentielle Kante zwischen zwei gegebenen Knoten in G enthalten, ist sie mit Wahrscheinlichkeit p nicht im erzeugten Zufallsgraphen. Und existierte keine Kante zwischen diesen beiden Knoten in G, wird mit Wahrscheinlichkeit p eine solche im erzeugten Zufallsgraphen hinzugefügt.

Sei $T(G)$ die (erwartete) Laufzeit des (probabilistischen) Algorithmus A, der das Graph-Problem auf dem Eingabe-Graphen G löst, und $T(RG(G,p))$ die erwartete Laufzeit dieses Algorithmus bei der Verteilung der Eingabe-Graphen gemäß $RG(G,p)$. Der Algorithmus A habe dann die *Smoothed-Komplexität* von $SC(n,p)$, wenn

$$SC(n,p) := \sup_{G \in \mathcal{G}_n} T(RG(G,p)) \tag{2.1}$$

gilt.

Dass dieser Begriff Eigenschaften des Algorithmus zwischen einer Worst- und einer Average-Case-Analyse aufzeigt, sieht man folgendermaßen: Einerseits erhält man bei der Wahl von $p = \frac{1}{2}$ Eingabe-Graphen, die uniform aus der Menge aller Graphen mit n Knoten gezogen werden. Damit erhält man mit dem Supremum die gleiche erwartete Laufzeit und diese Smoothed Analyse ist eine reine Average-Case-Betrachtung. Andererseits erhält man jedoch für $p = 0$ als Wahrscheinlichkeitsraum $RG(G,p)$ die Ein-Punkt-Verteilung, in welcher der Graph G mit Sicherheit gezogen wird. Die Bildung des Supremums liefert dann also eine reine Worst-Case-Aussage.

Das bisher berühmteste und klassische Resultat geben Spielman und Teng [95] direkt selbst an: Während der Simplex-Algorithmus zum Lösen linearer Optimierungsprobleme zwar in der Praxis sehr schnell die gesuchte Lösung ermittelt, konnte in [66] gezeigt werden, dass er eine exponentielle Worst-Case-Laufzeit besitzt. Jedoch sind diese „schwierigen“ Instanzen so selten, dass selbst eine kleine Störung der Eingabedaten ausreicht, um im Erwartungswert wieder eine polynomielle Laufzeit zu erhalten. Dieses bedeutende Ergebnis der Smoothed-Analyse konnte in [95] gezeigt werden. Seitdem wurde dieser Ansatz auch genutzt, um diskrete Probleme zu studieren, wie dies z. B. in [6, 13, 20, 28, 47, 69] geschehen ist.

2.5 Approximations-Algorithmen

Viele Optimierungs-Probleme der Theoretischen Informatik, die in praxisrelevanten Zusammenhängen in Erscheinung treten, gehören der Komplexitätsklasse NP an. Damit gibt es für diese keine Algorithmen, die in Polynomialzeit eine entsprechende Optimal-Lösung finden, es sei denn, es ist P = NP, was weithin als nicht der Fall angenommen wird.

Dabei sind mit einer solchen negativen Antwort, ein Problem ist NP-vollständig und lässt sich also nicht effektiv lösen, noch bei Weitem nicht alle Fragen geklärt. Verschiedene Ansätze, dennoch mit vertretbarem Zeitaufwand zu einer Lösung zu gelangen, werden wir mit einer parametrisierten bzw. probabilistischen Sichtweise in den nachfolgenden Kapiteln betrachten. Eine weitere mögliche Form der Relaxierung der Aufgabenstellung ist es, sich nicht unbedingt nach der Suche nur des jeweiligen Optimums zu machen, sondern auch Näherungslösungen zuzulassen, die bestenfalls nur wenig schlechter als das Optimum sind, sofern eine solche in polynomieller Zeit gefunden werden kann. Wenn man garantieren kann, schnell eine Lösung zu finden, die z. B. höchstens um einen Faktor $1+\varepsilon$ schlechter ist als das Optimum, dann kann dies durchaus völlig ausreichend für die praktischen Belange sein.

Ob ein Optimierungsproblem solche Approximations-Algorithmen zulässt, ist ein eigener, weiter Bereich in der Theoretischen Informatik. Dazu wurden viele Untersuchungen betrieben. Einen guten Überblick dazu findet man in [52].

Das wesentliche Maß zur Einschätzung der Güte eines Approximations-Algorithmus ist der durch ihn garantierte *Approximations-Faktor*. Da wir im Folgenden nur Minimierungsprobleme (d. h. Probleme, bei denen eine möglichst kleine Lösung gesucht ist) betrachten, geben wir die Definition nur für solche an.

Definition 2.5.1. Gegeben sei ein Approximations-Algorithmus A für das Minimierungsproblem P, d. h. insbesondere, dass A polynomielle Laufzeit besitzt. Für jede Instanz x von P sei $OPT(x)$ die Größe einer optimalen Lösung und $A(x)$ die Größe der von Algorithmus A ermittelten Lösung. Gilt nun für alle Instanzen x von P, dass $A(x) \leq r \cdot OPT(x)$ ist, so sagt man, dass der Algorithmus A eine r-Approximation garantiert bzw. einen Approximationsfaktor von (höchstens) r besitzt und damit das Problem P (durch A) r-approximierbar ist.

Die einfachste Sorte von Approximationsalgorithmen sind Greedy-Algorithmen: Dabei wird sukzessive das jeweils lokal beste Element ausgewählt, bis man eine Lösungsmenge erhält. Für viele Probleme, die einer Matroid-Struktur unterliegen, liefern diese Greedy-Algorithmen auch garantiert schon eine Optimal-Lösung. Ein Beispiel dafür ist der Algorithmus von Kruskal zur Bestimmung eines minimalen Spannbaums. Jedoch gehören auch viele Probleme nicht dieser Klasse an, und somit erhält man dort mit den Greedy-Algorithmen nur Näherungslösungen. Dennoch haben sie aufgrund ihrer geringen Laufzeit ihre Berechtigung, um schnell erste Näherungen zu berechnen. Wie gut diese sind, d. h., welcher Approximationsfaktor erreicht wird, wurde dabei auch schon für viele Probleme untersucht.

Auch bezüglich ihrer Approximierbarkeit kann man verschiedene Probleme wieder verschiedenen Komplexitätsklassen zuordnen. Die zwei wichtigsten Vertreter darunter sind APX und PTAS, die hier wie z. B. in [83], welches einen guten Übersichtsartikel zu Komplexitätsbetrachtungen bezüglich der Approximierbarkeit verschiedener Probleme darstellt, definiert werden:

Definition 2.5.2. Die Klasse aller derjenigen Probleme, für die es Approximationsalgorithmen mit höchstens konstantem Approximationsfaktor gibt, wird APX genannt. Besitzt ein solches Problem in APX ein Polynomialzeit-Approximations-Schema (engl. PTAS), d. h., dass es für jedes $\varepsilon > 0$ sogar $(1+\varepsilon)$-approximierbar ist, so ist es der Komplexitätsklasse $\mathsf{PTAS} \subseteq \mathsf{APX}$ zugehörig.

Dabei gilt $\mathsf{P} \neq \mathsf{NP} \Rightarrow \mathsf{PTAS} \neq \mathsf{APX}$. Betrachtet man hierbei analog der Polynomialzeitreduktion eine PTAS-Reduktion, kann man auch analog der NP-Vollständigkeit den Begriff der APX-Vollständigkeit definieren. Und ähnlich der Aussage über NP-vollständige Probleme folgt also auch hier, dass ein APX-vollständiges Problem „wahrscheinlich“ nicht in PTAS liegt, also nicht beliebig genau (in polynomieller Zeit) approximiert werden kann. Ein Beispiel dafür ist etwa mit dem Vertex-Cover-Problem gegeben, welches wohl keine 1, 36-Approximation durch deterministische Approximationsalgorithmen zulässt, siehe [33]. In Kapitel 5 werden wir jedoch aufzeigen, dass es einen probabilistischen Approximations-Algorithmus gibt, der in erwarteter FPT-Zeit (in Abhängigkeit des Parameters der Größe einer optimalen Knotenüberdeckung und der gewünschten Approximationsgüte) für jeden Eingabegraphen beliebig genaue Näherungslösungen liefert.

2.6 Probabilistische Algorithmen

Abschließen möchten wir diese Auflistung von Möglichkeiten, wie man jenseits der Beschränkungen deterministischer Worst-Case-Betrachtungen nach effizienten Algorithmen suchen kann, mit einer kurzen Beschreibung randomisierter bzw. probabilistischer Algorithmen, die in ihrer Berechnung Zufalls-Elemente nutzen (und sich auf diese Weise in gewissem Maße der Fähigkeiten nicht-deterministischer Turing-Maschinen bedienen).

Eine gute theoretische Einführung findet sich unter anderem in [99], während algorithmische Hinweise z. B. in [77] und für deren Analyse hilfreiche wahrscheinlichkeitstheoretische Grundlagen in z. B. [38] nachlesbar sind.

Hierbei sollen deterministische Algorithmen durch die Benutzung von begrenzten Zufalls-Entscheidungen effizienter gestaltet werden. Ein Beispiel dafür ist die Komplexitätsklasse ZPP, deren Definition wir im Folgenden angeben.

Definition 2.6.1. Die Komplexitätsklasse ZPP besteht aus allen Sprachen L, für die eine nicht-deterministische Turing-Maschine existiert, die folgende Bedingungen erfüllt:

- Jede Eingabe $x \in L$ wird akzeptiert, jede Eingabe $x \notin L$ abgelehnt.
- Es existiert ein Polynom p, sodass die erwartete Laufzeit der Turing-Maschine für jedes $x \in \Sigma^*$ durch $p(|x|)$ nach oben beschränkt ist.

Dabei steht ZPP für „zero-error probabilistic polynomial time“, d. h., der Algorithmus darf sich zwar aufgrund zufälliger Werte verschieden verhalten, muss aber immer die korrekte Antwort geben; und dies in erwarteter polynomieller Zeit. Solche Algorithmen, die definitiv die korrekte Antwort zurückgeben, heißen auch *Las-Vegas-Algorithmen.*

Während diese Algorithmen immer korrekt, aber nur zumeist schnell sein müssen, kann man die Anforderungen auch vertauschen: Immer schnell, aber nur zumeist korrekt. Dann erhält man *Monte-Carlo-Algorithmen.* Auch diese finden sich in Komplexitätsklassen wieder, je nachdem, ob man nur einen ein- oder auch einen zwei-seitigen Fehler zulässt:

Definition 2.6.2. Die Komplexitätsklasse RP besteht aus genau den Sprachen L, für die eine nicht-deterministische Turing-Maschine existiert, die folgende Bedingungen erfüllt:

- Auf jedem Berechnungspfad ist die Laufzeit polynomiell in der Eingabelänge beschränkt.
- Ist die Eingabe $x \notin L$, so wird sie auf jeden Fall abgelehnt.
- Ist die Eingabe $x \in L$, so wird sie mit Wahrscheinlichkeit von mindestens $\frac{1}{2}$ angenommen.

Analog besteht die Komplexitätsklasse BPP aus genau den Sprachen L, für die eine nicht-deterministische Turing-Maschine existiert, die folgende Bedingungen erfüllt:

- Auf jedem Berechnungspfad ist die Laufzeit polynomiell in der Eingabelänge beschränkt.
- Ist die Eingabe $x \notin L$, so wird sie mit Wahrscheinlichkeit von mindestens $\frac{2}{3}$ abgelehnt.
- Ist die Eingabe $x \in L$, so wird sie mit Wahrscheinlichkeit von mindestens $\frac{2}{3}$ angenommen.

Die Wahrscheinlichkeit für einen Fehler kann dabei in beiden Fällen durch Wiederholen unter jede beliebig kleine (aber positive) Schranke gedrückt werden: Sie sinkt exponentiell mit der Anzahl der wiederholten Anwendung des jeweiligen Algorithmus.

Wir bemerken aber, dass die in Kapitel 3 und 4 betrachteten probabilistischen Algorithmen Las-Vegas-Algorithmen sind, d. h. immer die korrekte Antwort ermitteln, während die in Kapitel 5 vorgestellten, probabilistischen Approximationsalgorithmen in gewissem Sinne als Monte-Carlo-Algorithmen gelten können, da sie jeweils nur mit bestimmter Wahrscheinlichkeit eine Lösung mit entsprechender Approximationsgüte liefern.

Zwischen den hier angegebenen Komplexitätsklassen gelten eine Reihe an Beziehungen. So ist offensichtlich $\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{BPP}$. Ob $\mathsf{BPP} \subseteq \mathsf{NP}$ gilt, ist ungeklärt. Da jedoch kaum Beispiele für Probleme, die in BPP liegen, von denen aber nicht bekannt ist, ob sie gleichzeitig auch Element von P sind, bekannt sind (und von einigen solcher früherer Beispiele schließlich die Zugehörigkeit zu P gezeigt werden konnte), vermutet man $\mathsf{P} = \mathsf{BPP}$. Das berühmteste Problem, dessen Zugehörigkeit zu BPP lang bekannt war, aber erst später die zu P gezeigt werden konnte, ist wohl Primes, welches fragt, ob eine gegebene natürliche Zahl eine Primzahl ist. Mittels des Miller-Rabin-Tests mit zufälligen Basen kann in erwarteter sehr schneller Zeit ein Zeuge für die Primalität bzw. Zusammengesetztheit der Eingabezahl gefunden werden. Jedoch erst der AKS-Primzahltest aus dem Jahr 2002 (veröffentlicht 2004, [2]) gab einen deterministischen Algorithmus in polynomieller Zeit (in Abhängigkeit der Eingabelänge, also dem Logarithmus der zu testenden Zahl) an.

Abschließend bemerken wir, dass der Einsatz probabilistischer Algorithmen auch im Bereich der parametrisierten Komplexität fruchtbare Ergebnisse gezeigt hat. So kann man einen randomisierten Algorithmus für das k-Path-Problem angeben, der in erwarteter FPT-Zeit läuft. Dabei besteht das Problem in der Frage, ob es in einem gegebenen Graphen einen Pfad der Länge $\geq k$ gibt. Jedoch kann man durch Derandomisierungs-Techniken aus diesem probabilistischen wieder einen deterministischen Algorithmus erzeugen, der weiterhin in FPT-Zeit läuft, [37].

2.7 Zufallsgraph-Modelle

Möchte man über das Verhalten von Algorithmen bei „zufälligen“, „durchschnittlichen“ oder „verrauschten“ Eingabedaten Aussagen machen, so hängen diese immer von der dabei vorausgesetzten Verteilung der Eingabedaten ab: Gemäß einer vorgegebenen Zufallsverteilung wird eine bestimmte Instanz gezogen, auf welche dann der betrachtete Algorithmus angewendet wird.

Da wir im Folgenden hauptsächlich graphentheoretische Problemstellungen betrachten werden, sollen hier einige Modelle für Zufallsgraphen, die also solche Verteilungen liefern, vorgestellt werden.

2.7.1 Erdős-Rényi Zufalls-Graphen

Ein sehr bekanntes Zufallsgraphen-Modell wurde von Erdős und Rényi entwickelt, [39], und ist wie folgt definiert:

Definition 2.7.1 (Erdős-Rényi Zufalls-Graphen)**.** Sei n eine positive, ganze Zahl und $p(n)$ eine Funktion $\mathbb{N} \to [0,1]$. Dann ist $G(n,p)$ der Zufallsraum auf der Menge der Graphen mit n Knoten, wobei jede Kante in einem daraus zu ziehenden Graphen mit Wahrscheinlichkeit $p = p(n)$ vorhanden ist und die Anwesenheit je zweier solche Kanten dabei unabhängig voneinander ist.

In [17] und [39] werden auch eine Reihe von Eigenschaften diskutiert, wie diese sich wandeln, wenn man sie während des Prozesses der kontinuierlichen Erhöhung der Kantenwahrscheinlichkeit von 0 bis 1 betrachtet. So kann man etwa bezüglich der Frage des Zusammenhangs die folgenden Aussagen treffen (die jeweils alle unter der Einschränkung „mit hoher Wahrscheinlichkeit“ gelten, d. h., dass deren Eintrittswahrscheinlichkeit mit steigender Knotenanzahl gegen 1 geht):

- Ist $p < \frac{1}{n}$, so besitzen alle Zusammenhangskomponenten des Graphen jeweils nur die Größe $\mathcal{O}(\log n)$.
- Für $p = \frac{1}{n}$ wächst die größte Zusammenhangskomponente auf $\mathcal{O}\left(n^{\frac{2}{3}}\right)$ Knoten an.
- Für $\frac{1}{n} < p \leq \frac{c}{n}$ mit einer Konstanten $c > 1$ enthält die größte Zusammenhangskomponente schließlich einen positiven Anteil aller Knoten des Graphen. Die weiteren Zusammenhangskomponenten besitzen wieder nur eine Größe von $\mathcal{O}(\log n)$.
- Ist $p < \frac{(1-\varepsilon)\log n}{n}$ mit einem $\varepsilon > 0$, so besitzt der Graph isolierte Knoten, ist also insbesondere nicht zusammenhängend.
- Ist dagegen $p > \frac{(1+\varepsilon)\log n}{n}$ mit einem $\varepsilon > 0$, so ist der Graph schließlich zusammenhängend.

Ähnliche Beschreibungen sind auch für andere Graph-Parameter wie etwa die Größe der größten Clique und weitere möglich.

Zu bemerken ist noch, dass $G\left(n, \frac{1}{2}\right)$ die Gleichverteilung über alle Graphen mit n Knoten darstellt.

2.7.2 Skalenfreie Graphen

Hier möchten wir Zufallsgraph-Modelle vorstellen, die im Gegensatz zu Erdős-Rényi-Graphen inhomogen sind, d. h., eine sehr ausdifferenzierte Gradverteilung (wenige Knoten mit hohem und viele Knoten mit niedrigem Grad) besitzen. Motivation für die Beschäftigung mit solchen Graphen ist, dass sie in vielen Bezügen der realen Welt wiederzufinden sind, da sie eine gewisse „Selbstähnlichkeit“ besitzen. So lassen sich z. B. die Zusammenhangsstruktur Sozialer Netzwerke, die Verbindungen großer Internet-Knoten oder auch Kollaborations-Graphen verschiedener Professionen (wie z. B. auch im Filmgeschäft, für welches man etwa das Zusammenwirken verschiedener Darsteller aus den Daten der Internet-Filmdatenbank IMDb erhalten kann) gut mit derartigen Strukturen abbilden.

Das Konzept der *Skalenfreiheit* besagt dabei, dass sich das Netzwerk in „selbstähnlichen“ Strukturen organisiert. So gibt es dabei nur eher wenige Knoten mit hohem Grad. Betrachtet man aber nur diese „Hubs“, so folgt auch in diesem Subgraphen die Gradverteilung dem gleichen Verteilungsgesetz: Es gibt einige, wenige Knoten hohen Grads und viele mit kleinem Grad.

In der Folge möchten wir einige dieser Modelle nur sehr kurz vorstellen, da sie in der weiteren Arbeit hier keine große Rolle spielen werden. Der Vollständigkeit des Themenkomplexes wegen sollen sie aber dennoch aufgeführt werden.

Preferential-Attachment-Graphen

Das berühmteste dieser Modelle stammt von Albert und Barabási [5]. Diese gehen von einem „rich get richer“-Ansatz aus: Der Graph wird sukzessive, von einem kleinen Kern beginnend, durch Hinzukommen neuer Knoten wie folgt konstruiert: Für jeden bisher bereits vorhandenen Knoten wird eine Kante zum neu hinzukommenden Knoten mit einer Wahrscheinlichkeit, die proportional zu seinem bisherigen Grad ist, gezeichnet. Dadurch erhalten die Knoten, die schon bisher einen hohen Grad haben, mit höherer Wahrscheinlichkeit neue Nachbarn als Knoten, deren Grad auch zuvor klein war.

Dieser Ansatz folgt der Überlegung, dass – am Beispiel eines Sozialen Netzwerks – eine neu hinzugekommene Person sich mit höherer Wahrscheinlichkeit mit populären Personen, also solchen, die schon sehr viele Follower besitzen, verbindet, als einem der vielen „No Names“.

Die Gradverteilung ist in diesem Modell proportional zur Funktion k^{-3}, d. h., der Anteil der Knoten vom Grad k im gesamten Netzwerk ist proportional zu jener polynomiell abnehmenden Funktion. Man sagt auch, die Gradverteilung folgt einem „power law“, einem „Potenz-Gesetz“.

Chung-Lu-Graphen

Die Schwierigkeit beim Arbeiten mit Preferential-Attachment-Graphen ist die durch die schrittweise Konstruktion gegebene Abhängigkeit der Kanten untereinander. Das Modell, welches in [3] vorgeschlagen wird, hat dieses Problem nicht. Hier werden alle Kanten gleichzeitig erzeugt:

Jedem Knoten wird hierfür ein positives Gewicht zugeordnet, wobei diese Gewichte selbst schon einer „Potenz-Gesetz“-Verteilung folgen. Eine Kante wird nun zwischen zwei Knoten genau mit einer Wahrscheinlichkeit proportional zum Produkt ihrer beiden Gewichte hinzugefügt. Dies sichert, dass der erwartete Grad eines Knotens proportional zu seinem Gewicht ist.

In [21] konnten die dortigen Autoren mittels dieses Modells zeigen, dass entsprechend aufgebaute Soziale Netzwerke sich leicht de-anonymisieren lassen, d. h., sich ein Isomorphismus zwischen zwei Versionen des gleichen Graphen finden lässt und man so aus den Zusatzinformationen aus der einen auf weitere Informationen aus der anderen schließen kann.

Hyperbolische Zufallsgraphen

Auch bei diesem Modell geht es darum, zu erklären, wie sich in Netzwerken eine solche Verteilung mit wenigen sehr starken und vielen schwachen Knoten herausbildet. Bei hyperbolischen Zufallsgraphen geht man davon aus, dass die Knoten sich durch eine zufällige Punktmenge in der hyperbolischen Ebene darstellen lassen. Die Kanten zwischen ihnen entstehen nun nach einem Abstands-Argument: Je näher sich die beiden Knoten in der hyperbolischen Ebene liegen, desto größer ist die Wahrscheinlichkeit für eine Kante zwischen ihnen. Auf diese Weise erhält man also „unter“ der reinen Graphen-Struktur eine zusätzliche, geometrische Information, die dann den Graphen und dessen Aussehen erst erzeugt.

In [84] zeigen die dortigen Autoren zum Beispiel, dass sie den „Internet-Graph“, der die Verbindungen zwischen den großen Backbones darstellt, sehr gut als einen solchen hyperbolischen Graphen auffassen können: Sie gewinnen dadurch die „darunter liegenden“ geometrische Information zurück und können dies dazu nutzen um zu zeigen, dass ein damit arbeitendes Greedy-Routing von Informationspaketen, die durch das Netzwerk gesendet werden sollen, schnell, effektiv und robust arbeitet.

2.7.3 Zufalls-Schnittgraphen

Eine weitere Möglichkeit, Zusammenhänge zwischen verschiedenen Objekten zu betrachten, sind Schnittgraphen. Dabei besitzt jeder Knoten eine gewisse Menge an Attributen und eine Kante existiert zwischen zwei Knoten genau dann, wenn sie mindestens ein gemeinsames Attribut besitzen. In einer alternativen Sicht weist man jedem Knoten einen Binärvektor zu (wobei dessen i-te Komponente codiert, ob der Knoten das i-te Attribut besitzt oder nicht) und verbindet zwei Knoten, wenn es mindestens eine Komponente gibt, für die beide zugehörigen Vektoren den Eintrag 1 besitzen. Wählt man dabei diese Vektoren zufällig, ergibt sich ganz automatisch ein entsprechendes Zufallsgraph-Modell:

Definition 2.7.2 (Zufalls-Schnittgraphen)**.** Sei $n > 0$ eine ganze Zahl. Wir konstruieren den Zufallsraum $CC_k(n, p)$ auf der Menge der Graphen mit n Knoten wie folgt:

- Sei $q := \left(1 - (1-p)^{\frac{1}{k}}\right)^{\frac{1}{2}}$.
- Für jeden Knoten v wähle unabhängig und zufällig einen Vektor $c_v \in \{0,1\}^k$.
- Dabei sei jeder Eintrag unabhängig voneinander 1 mit Wahrscheinlichkeit q und sonst 0.
- Füge nun eine Kante zwischen zwei verschiedenen Knoten v und w genau dann hinzu, wenn ein Index $1 \leq \ell \leq k$ existiert, sodass beide Vektoren c_v und c_w einen Eintrag 1 an diesem Index ℓ besitzen.
- Alle anderen Graphen erhalten die Wahrscheinlichkeit 0.

Zufalls-Schnittgraphen wurden schon in verschiedenen Zusammenhängen betrachtet z. B. durch Michal et al. [76] für Teilgraphen und durch Behrisch und Taraz [8] für Clique-Überdeckungen. Für einen Überblick aktueller Ergebnisse zu Zufalls-Schnittgraphen ist Zhao et al. [101] eine gute Quelle.

Es ist darauf hinzuweisen, dass die übliche Definition von Zufalls-Schnittgraphen die „Vektor-Wahrscheinlichkeit“ q direkt nutzt. Unsere auf der Kantenwahrscheinlichkeit p fußende Definition ist dazu äquivalent: Für gegebenes k erhält man aus Definition 2.7.2 eine direkte Umrechnung von p zu q und umgekehrt. Um einen besseren Vergleich mit dem „Standard“-Zufallsgraph-Modell $G(n, p)$ mit Kantenwahrscheinlichkeit $p(n)$ von Erdős und Rényi zu haben, bevorzugen wir jedoch diese Form der Definition der Zufalls-Schnittgraphen.

Das Modell der Zufalls-Schnittgraphen verhält sich sehr ähnlich zu Erdős-Rényi-Graphen, was die Verteilung der Knotengrade betrifft:

Satz 2.7.3. *Seien $n > 0$, $0 < p < 1$ und $CC_k(n,p)$ der oben definierte Zufallsraum der Zufalls-Schnittgraphen. Sei G zufällig gemäß $CC_k(n,p)$ ausgewählt. Dann gelten folgende Eigenschaften:*

- *Für alle potentiellen Kanten ist die Auftrittswahrscheinlichkeit p. Ihr Auftreten ist jedoch nicht unabhängig voneinander.*
- *Der erwartete Grad eines jeden Knotens ist gleich. Damit handelt es sich um ein homogenes Zufallsgraph-Modell.*

Beweis. Man betrachte zwei verschiedene Knoten. Eine Kante existiert zwischen diesen genau dann, wenn an mindestens einer der k Stellen beide zugehörigen Vektoren den Eintrag 1 besitzen. Das Gegenereignis wäre also, dass in allen k Komponenten keine zwei Einsen zu finden sind. In jeder Komponente geschieht dies mit Wahrscheinlichkeit $1 - q^2 = (1-p)^{\frac{1}{k}}$, d. h. gleichzeitig in allen mit Wahrscheinlichkeit $\left((1-p)^{\frac{1}{k}}\right)^k = 1 - p$, da die Komponenten unabhängig voneinander sind. Damit hat also jede potentielle Kante zwischen je zwei verschiedenen Knoten in diesem Graphen die Wahrscheinlichkeit p.

Und da die Definition invariant unter einer Permutation der Knoten ist, besitzen alle Knoten den gleichen erwarteten Knotengrad. □

2.7.4 Allgemeine Zufallsgraphen und Zufalls-Mengen-Familien

In Kapitel 4 werden wir ein recht allgemeines Zufallsgraph-Modell betrachten. Dazu nutzen wir den gleichen, allgemeinen Ansatz wie Lu und Peng [73]. In dieser Arbeit wurden die Spektral-Eigenschaften der Adjazenz-Matrizen solcher Zufallsgraphen untersucht. Die einzige, wesentliche Annahme dieses Zufallsgraph-Modells ist die totale Unabhängigkeit der Kanten:

Definition 2.7.4. Sei $1 < n \in \mathbb{N}$ und $P := (p_{i,j})_{1 \leq i,j \leq n} \in [0,1]^{n \times n}$ eine symmetrische Matrix mit 0-Einträgen auf der Hauptdiagonalen. Den Wahrscheinlichkeitsraum $RG(n,P)$ von Graphen mit n Knoten definieren wir nun durch das Ziehen der Kanten zwischen zwei Knoten $1 \leq i < j \leq n$ mit Wahrscheinlichkeit $p_{i,j}$, wobei alle diese Züge unabhängig voneinander sind.

Gibt es dabei Schranken $0 \leq p \leq q \leq 1$ mit $p \leq p_{i,j} \leq q$ für alle $1 \leq i < j \leq n$, nennen wir die Wahrscheinlichkeits-Matrix P (p,q) - *beschränkt* und fassen alle diese Wahrscheinlichkeitsräume in der Menge $RG(n,p,q)$ zusammen. Ist $q = 1$, wird es nicht notiert.

Dabei dürfen p und q von n abhängen. Zusammenfassend bedeutet die Aussage „Sei G ein gemäß $RG(n,p)$ gezogener Zufallsgraph“, kurz $G \sim RG(n,p)$, dass die dann im folgenden angegebenen Sätze und Korollare für alle p-beschränkten Wahrscheinlichkeits-Matrizen P und dem jeweils dazugehörigen Wahrscheinlichkeitsraum $RG(n,P)$ gelten.

Diese Definition von Zufallsgraphen kann als Verallgemeinerung von Erdős-Rényi-Zufallsgraphen, wie sie in [39] betrachtet werden, aufgefasst werden. Beim dortigen Modell besitzen alle potentiellen Kanten die gleiche Wahrscheinlichkeit, im Graphen enthalten zu sein (d. h. $G(n,p) = RG(n,p,p)$). Auch verallgemeinert das hier angegebene Zufallsgraph-Modell das von Chung-Lu-Zufallsgraphen, wie sie in [3] betrachtet werden, bei welchen die Wahrscheinlichkeit einer Kante zwischen zwei Knoten proportional zum Produkt der Gewichte ihrer Endknoten ist. Typischerweise nehmen wir $p, q \in (0,1)$ an, um garantierte (Nicht-)Kanten zu vermeiden. Würden diese Wahrscheinlichkeiten z. B. nur um höchstens $1/n^2$ von 0 bzw. 1 abweichen, könnte man damit mit hoher Wahrscheinlichkeit eine Konzentration um eine Worst-Case-Instanz erzeugen.

Auf analoge Weise definieren wir auch Zufalls-Mengen-Familien, die bei der Betrachtung des (d)-Hitting-Set-Problems eine Rolle spielen werden:

Definition 2.7.5. Sei $1 \leq n \in \mathbb{N}$, G eine n-elementige Menge mit Potenzmenge $\mathcal{P}(G)$ und $P : \mathcal{P}(G) \to [0,1], S \mapsto P(S)$ eine Wahrscheinlichkeitsfunktion, die jeder Teilmenge S von G eine individuelle Auftrittswahrscheinlichkeit $p(S)$ zuordnet. Der Wahrscheinlichkeitsraum $RG(n,P)$ von Mengenfamilien über Grundmengen mit n Elementen wird im Folgenden definiert durch das Ziehen der einzelnen potentiellen Teilmengen S von G mit Wahrscheinlichkeit $p(S)$, wobei alle diese Züge unabhängig voneinander erfolgen.

Gibt es dabei Schranken $0 \leq p \leq q \leq 1$ mit $p \leq p(S) \leq q$ für alle $S \subset G$, nennen wir die Wahrscheinlichkeitsfunktion P (p,q)-*beschränkt* und fassen alle diese Wahrscheinlichkeitsräume in der Menge $RS(n,p,q)$ zusammen. Sind für ein $d \in \mathbb{N}$ und alle Teilmengen der Größe genau gleich d die Einzelwahrscheinlichkeiten (p,q)-beschränkt, nennen wir diese Teilmenge der Wahrscheinlichkeitsräume $RS_d(n,p,q)$.

Wir bemerken, dass dabei dieses Modell zufälliger Mengen-Familien das vorgenannte Zufallsgraph-Modell verallgemeinert. Tatsächlich ist nämlich $RG(n,p,q) = RS_2(n,p,q)$, wobei im letzteren alle Wahrscheinlichkeiten für Mengen mit verschieden von zwei Elementen auf 0 gesetzt wurden: Die Grundmenge G steht dabei für die Knotenmenge des Zufallsgraphen und jede Zweiermenge dafür, ob die Kante zwischen den beiden entsprechenden Knoten im Graphen (und so der Mengenfamilie) enthalten ist oder nicht. Man kann das allgemeine Mengen-Modell somit auch als Hypergraph betrachten.

Wie auf die Definition 2.7.4 folgend, wollen wir auch hier für die Aussage „Sei $\mathcal{F}$ eine gemäß $RS(n,p,q)$ gezogene Zufalls-Mengenfamilie“ kurz $\mathcal{F} \sim SG(n,p,q)$ schreiben und im Fall $q = 1$ dieses nicht weiter notieren. Genauso gehen wir auch bei den Wahrscheinlichkeitsräumen $SG_d(n,p,q)$ vor. Weiterhin dürfen dabei auch d sowie p und q von n abhängen.

2.8 Wahrscheinlichkeitstheoretische Grundlagen

2.8.1 Konzentration um Erwartungswert und Chernoff-Schranken

In dieser Arbeit werden häufiger Wahrscheinlichkeitsaussagen benötigt, wie weit die Summe von bestimmten Zufallsvariablen von ihrem Erwartungswert abweichen kann. Eine gut zu verwendende obere Abschätzung für die Wahrscheinlichkeit einer „großen“ Abweichung gibt die Chernoff-Schranke, die wir wie in Dubhashi und Panconesi [38] angeben.

Lemma 2.8.1 (Chernoff-Schranke)**.** *Sei X die Summe unabhängiger Zufallsvariablen, deren Werte in $\{0,1\}$ liegen, μ der Erwartungswert von X und $0 < \delta < 1$ eine reelle Zahl. Dann ist*

$$P\left(X \leq (1-\delta)\mu\right) \leq \exp\left(-\frac{\delta^2}{2}\mu\right) \text{ und } P\left(X \geq (1+\delta)\mu\right) \leq \exp\left(-\frac{\delta^2}{3}\mu\right).$$

2.8.2 „Inverse“ Chernoff-Schranke

Anders als durch die in der Chernoff-Schranke gegebene obere Abschätzung, wie wahrscheinlich höchstens ein gewisses Abweichen vom Erwartungswert ist, benötigen wir zum Teil auch die umgekehrten Aussagen, wie wahrscheinlich es mindestens ist. Solche „inversen“ Chernoff-Schranken gibt es in verschiedenen Formulierungen, die sich nur in Konstanten von der in Lemma 2.8.1 unterscheiden. Wir folgen dabei inhaltlich der Formulierung von [93] und[78] und geben sie mit den Bezeichnungen an, wie wir sie auch weiterhin in dieser Arbeit verwenden:

Lemma 2.8.2 (Inverse Chernoff-Schranke)**.** *Sei X die Summe unabhängiger und gleichverteilter Zufallsvariablen X_i, deren Werte in $\{0,1\}$ liegen, $P[X_i = 1] = p$, μ der Erwartungswert von X und $0 < \delta$ eine reelle Zahl. Ist zusätzlich $p \leq \frac{1}{4}$ oder zugleich sowohl $p < \frac{1}{2}$ als auch $\delta \leq \frac{1}{p} - 2$, dann ist*

$$P\left(X \geq (1+\delta)\mu\right) \geq \frac{1}{4}\exp\left(-2\delta^2\mu\right).$$

3 Kernelisierung am Beispiel des Clique-Cover-Problems

Die Ergebnisse, die in diesem Abschnitt vorgestellt werden, wurden vom Autor und seinem Betreuer in [44] bereits veröffentlicht.

3.1 Überblick

Die Überdeckung aller Kanten eines Graphen durch eine minimale Anzahl von Cliquen ist ein bekanntes NP-vollständiges Problem. Wählt man als Parameter k die maximale Anzahl von benötigten Cliquen, so wird das Problem „fixed parameter tractable“, d. h., es existiert ein dieses Problem lösender Algorithmus mit Laufzeit $f(k) \cdot n^{\mathcal{O}(1)}$. Genauer: Es existiert eine Menge von Reduktionsregeln, die für alle Instanzen, die sich mit k Cliquen überdecken lassen, einen Problemkern der Größe höchstens 2^k erhalten. Jedoch gibt es unter der Annahme der Exponentialzeit-Hypothese keine solche Regelmenge, die auch im Worst-Case subexponentiell große Problemkerne erzeugt.

Wir studieren die erwartete Problemkerngröße für „Zufalls-Schnittgraphen“ mit n Knoten, Kantenwahrscheinlichkeit p und Clique-Überdeckungen der Größe k. Dabei betrachten wir die bekannte Menge an Reduktionsregeln von Gramm et al. [54] und zeigen, dass mit hoher Wahrscheinlichkeit diese den Graphen vollständig reduzieren, falls p nicht gegen 1 geht und $k < c \log n$ für eine Konstante $c > 0$ ist. Dies zeigt, dass für große Klassen von Graphen wie „Zufalls-Schnittgraphen“ die erwartete Größe des Problemkerns deutlich kleiner als die bekannte, exponentiellele Worst-Case-Schranke sein kann.

3.2 Einführung

In den vergangenen Jahren wurden einige Resultate zu oberen und unteren Schranken der Größe von Problemkernen parametrisierter Probleme gezeigt. Nahezu alle von diesen betrachten nur den Worst-Case. Wir verfolgen einen anderen Ansatz und betrachten die erwartete Größe des Problemkerns für ein probabilistisches Zufallsgraph-Modell und geben dafür eine scharfe Charakterisierung in Abhängigkeit der Dichtheit der Graphen an.

Unser Studienobjekt ist das NP-vollständige Problem Clique-Cover, bei welchem alle Kanten eines Graphen mit einer minimalen Anzahl von Cliquen überdeckt werden sollen. Dieses Problem stellt sich z. B. bei der Untersuchung der Interaktion

von Entitäten in Netzwerken der realen Welt [56] und Protein-Protein-Interaktions-Netzwerken [12]. Weitere Anwendung findet es in der Computer-Geometrie [1], der Optimierung von Compilern [89] und Computer-Statistik [86, 53].

In verschiedenen Bereichen wurde dieses Problem durch unterschiedliche Namen beschrieben, wie z. B. Schlüsselwort-Konflikt [68], Überdeckung durch Cliquen [51] und Zufalls-Schnittgraphen-Basis [50].

Formale Definition des Problems Eine Clique in einem ungerichteten Graphen $G = (V, E)$ ist ein Subgraph, in welchem je zwei Knoten durch eine Kante verbunden sind. Für einen ungerichteten Graphen G und eine ganze Zahl $k \geq 0$ lautet die Antwort auf das Entscheidungsproblem (Edge) Clique-Cover genau dann „Ja“, wenn es eine Menge von maximal k Cliquen $\{G_1, G_2, \ldots, G_k\}$ in G gibt, sodass für jede Kante in G ihre beiden Endpunkte gleichzeitig in mindestens einer dieser Cliquen G_i liegen.

Bisher gefundene Resultate für spezielle Graph-Klassen Das Problem Clique-Cover wurde schon aus verschiedensten Richtungen betrachtet. Es bleibt NP-schwer, selbst wenn man sich auf planare Eingabe-Graphen [24] oder solche mit Maximal-Grad 6 [60] beschränkt. Bei Graphen mit Maximal-Grad 5 gibt es jedoch Polynomialzeit-Algorithmen [60], ähnlich bei Chordal-Graphen [75] oder Linien-Graphen [82]. Wieder im allgemeinen Fall zeigen Lund und Yannakakis, dass die Optimierungs-Variante von Clique-Cover nicht mit einem Faktor von $|V|^{\varepsilon}$ für ein $\varepsilon > 0$ approximierbar ist (es sei denn P = NP) [74]. Genauer formuliert, bleibt es APX-schwer, selbst wenn man sich auf zweifach verbundene Graphen mit Maximal-Grad 7 beschränkt [59]. Daher ist kein guter und schneller Approximations-Algorithmus zu erwarten.

Bisherige Ergebnisse für das parametrisierte Problem Ein anderer, weit verbreiteter Ansatz, solche Probleme zu betrachten, ist das Studium seiner parametrisierten Version. Fixiert man die Größe der gewünschten Lösung als Parameter k, so findet man in [58] eine Menge einfacher Reduktionsregeln. Erfolgreiche Anwendung dieser Regeln führt dabei zu einem kleineren Graphen und/oder Parameter. Dieser Prozess wird *Kernelisierung* genannt, da er als Ergebnis einen sogenannten Problem-Kern liefert, welcher genau dann eine „Ja“-Instanz ist, wenn es der ursprüngliche Graph auch war.

Die in [58] angegebenen Reduktionsregeln erzeugen dabei einen Problemkern mit maximal 2^k Knoten. Diese Kernelisierung kann dabei in Polynomialzeit berechnet werden. Da die dabei erhaltene Kern-Größe nicht von $|V| = n$ abhängt, ist Clique-Cover mit dieser Parametrisierung also in FPT, vgl. Gramm et al. [54]. Führt man die angegebenen Reduktionsregeln aus, erhält man damit einen Algorithmus mit Laufzeit $\mathcal{O}\left(n^4 + f\left(2^k\right)\right)$. Dabei ist für die Funktion f nur der naive Ansatz

der vollständigen Aufzählung aller möglicher Lösungsmengen auf dem verbleibenden Problemkern bekannt. Dieser besitzt exponentielle Laufzeit in Abhängigkeit der Anzahl der Knoten im Problemkern, also doppelt-exponentielle in Abhängigkeit vom Parameter k.

Aktuelle Ergebnisse von Cygan et al. [32, 31] zeigen, dass es keine Reduktionsregeln geben kann, welche nur polynomiell große Kerne garantieren, wenn nicht $\mathsf{NP} \subset \mathsf{coNP}/\operatorname{poly}$ gilt, und dass es keine mit garantiert subexponentieller Kern-Größe gibt, es sei denn die Exponentialzeit-Hypothese ist falsch. Genauer formuliert, muss der Problemkern im Worst-Case exponentielle Größe haben, es sei denn $\mathsf{P} = \mathsf{NP}$, vgl. [87]. Mit diesen Negativ-Ergebnissen ist also kein Algorithmus für das Problem zu erwarten, der eine Laufzeit von $2^{2^{o(k)}} \cdot \operatorname{poly}(n)$ besitzt – doppelt exponentielle Laufzeit wird also benötigt. Dabei hängen diese Resultate nicht von der konkret verwendeten Menge von Reduktionsregeln ab. Generell sind dabei die Größen der Problemkerne (ob nun polynomiell oder nicht) von großem Interesse, da sie zu einer feineren Struktur-Analyse von FPT-Problemen führen, vgl. Bodlaender et al. [16].

Bisherige Resultate bezüglich einer Average-Case-Betrachtung Das Clique-Cover-Problem wurde auch auf Zufallsgraphen studiert. Bollobás et al. [18] geben obere und untere Schranken für die Anzahl der Cliquen an, die benötigt werden um den gesamten Graphen zu überdecken. Diese Schranken gelten dabei mit hoher Wahrscheinlichkeit (d. h. mit Wahrscheinlichkeit $1 - o(1)$ für $n \to \infty$) für Erdős-Rényi-Zufallsgraphen mit Kanten-Wahrscheinlichkeit $p = \frac{1}{2}$. Dies ist äquivalent zu einer Gleichverteilung aller Graphen mit der Knotenmenge $|V| = \{1, 2, \ldots, n\}$. Das Ergebnis wurde verbessert zu $\Theta(n^2/\log^2 n)$ für alle konstanten p mit $0 < p < 1$ durch Frieze und Reed [48]. Auch auf Schnittgraphen wurde das Problem betrachtet. Behrisch und Taraz [8] geben Algorithmen an, welche mit hoher Wahrscheinlichkeit eine Überdeckung durch eine minimale Anzahl an Cliquen in polynomieller Zeit finden, wenn die zugrunde liegende Eigenschaftsmenge (und damit die erwartete Größe des Clique-Cover) eine Größe von n^α für eine Konstante $0 < \alpha < 1$ besitzt.

Unsere Ergebnisse Wir untersuchen die erwartete Kern-Größe in einem Zufallsgraph-Modell mit Namen „Zufalls-Schnittgraphen“, welches ähnlich zu Erdős-Rényi ist, aber sichert, dass die erhaltenen Graphen jeweils mit höchstens k Cliquen überdeckt werden können. (Genaueres dazu in Abschnitt 3.4.) Zuerst untersuchen wir eine bestimmte Menge von Reduktionsregeln (die in Abschnitt 3.5 definiert werden), welche sich intuitiv aus der Definition des Zufallsgraph-Modells ergeben. In Abschnitt 3.6 zeigen wir, dass für dünne Graphen (bei welchen die Kantenwahrscheinlichkeit p mindest polynomiell gegen 0 abfällt) diese Reduktionsregeln mit hoher Wahrscheinlichkeit einen im Parameter k nur polynomiell großen Problemkern erzeugen. Demzufolge sind Worst-Case-Instanzen mit exponentiell großem Problemkern selten. Ist der Graph dicht (d. h., die Kantenwahrscheinlichkeit fällt nicht polynomiell auf 0, also z. B. für konstante p mit $0 < p < 1$), führt diese Menge an

Reduktionsregeln mit hoher Wahrscheinlichkeit die Instanzen nur auf einen exponentiell großen Problemkern zurück. Wir geben eine vollständige Charakterisierung der (mit hoher Wahrscheinlichkeit angenommenen) Kern-Größen in Abhängigkeit der Kantenwahrscheinlichkeit in Satz 3.6.1 an.

Diese stufenweise fallende Kern-Größe ist in Grafik 3.1 auf Seite 42 abgebildet. Bei Kantenwahrscheinlichkeiten $p(n)$, welche sich asymptotisch wie $n^{-2/i}$ mit $1 < i \leq k$ verhalten (genauer: für die

$$\lim_{n\to\infty} \frac{\log p}{\log n} = -\frac{2}{i} =: -a$$

gilt), kann das Verhalten der Kern-Größe nicht vollständig allein durch den Parameter a beschrieben werden und so ist nur das entsprechende Intervall angegeben, in welchem sich mit hoher Wahrscheinlichkeit die Kern-Größe bewegt.

In einem zweiten Schritt verallgemeinern wir die bisherige Menge an Reduktionsregeln und untersuchen die durch Gramm et al. [54] gegebene Regelmenge. Für diese zeigen wir in Abschnitt 3.7 mit den gleichen Techniken, dass für Kantenwahrscheinlichkeiten $p < 1 - r^k$ für eine Konstante $0 < r < 1$ und $k < c(r) \log n$ mit einer positiven Konstanten $c(r)$, die von r abhängt, diese Reduktionsregelmenge den Graphen mit hoher Wahrscheinlichkeit komplett reduziert, d. h., der Problemkern der leere Graph ist. Für Schnittgraphen kann damit die Menge an Reduktionsregeln, die durch Gramm et al. [54] angegeben werden, also als „optimale Regelmenge“ für eine Kernelisierung angesehen werden.

3.3 Erdős-Rényi Zufalls-Graphen

Die bekannte Beobachtung, dass das Finden einer minimalen Clique-Überdeckung NP-vollständig ist, ist eine reine Worst-Case-Aussage. Um das Problem besser zu verstehen, werden im Folgenden verschiedene Zufallsgraph-Modelle betrachtet und auf diesen für das Problem eine Average-Case-Analyse durchgeführt.

Als prominentestes und am meisten studiertes Zufallsgraph-Modell betrachten wir zuerst die Erdős-Rényi Zufalls-Graphen. Ihre Konstruktion ist in Definition 2.7.1 gegeben und es finden sich einige Eigenschaften in Abschnitt 2.7.1 dieser Arbeit. Auch das Verhalten des Clique-Cover-Problems auf Erdős-Rényi Zufalls-Graphen wurde unter anderem durch Bollobás et al. [18] und Frieze und Reed [48] betrachtet. Für eine konstante Kanten-Wahrscheinlichkeit $0 < p < 1$ geht mit hoher Wahrscheinlichkeit die Anzahl der benötigten Cliquen zur Überdeckung des Graphen mit der Knotenanzahl gegen unendlich. Halten wir demnach umgekehrt die Anzahl k solcher Cliquen, die für die Überdeckung zur Verfügung stehen, fest, so sind fast alle Instanzen „Nein“-Instanzen. Dies kann man auch exakter wie folgt formulieren.

Lemma 3.3.1. *Sei k eine fest gewählte, positive ganze Zahl, $G(n,p)$ der Zufallsraum gemäß des Erdős-Rényi-Modells mit Kantenwahrscheinlichkeit $p(n)$ und der Knotenanzahl n, die gegen unendlich strebe. Geht dabei nicht $p(n)$ gegen 1 und ist in $\omega\left(\frac{1}{n^2}\right)$, so ist mit hoher Wahrscheinlichkeit ein so gezogener Graph eine „Nein"-Instanz.*

Beweis. In einem aus dem Zufallsraum $G(n,p)$ mit nicht gegen 1 gehendem p gezogenen Graphen besteht mit hoher Wahrscheinlichkeit die größte Clique aus nur $\mathcal{O}(\log n)$ Knoten. Damit können also maximal $\mathcal{O}\left(k \cdot \log^2 n\right)$ Kanten durch k solche Cliquen überdeckt werden. Für $p = \omega(\log^2(n)/n^2)$ gibt es mehr zu überdeckende Kanten und also handelt es sich mit hoher Wahrscheinlichkeit bei dem Graph um eine „Nein"-Instanz. Ist dagegen $p = \mathcal{O}\left(\frac{\log^2 n}{n^2}\right) = o\left(\frac{1}{n}\right)$, so besitzt fast jeder Knoten einen Grad von höchstens 1. Um diese Kanten zu überdecken, bräuchte es also jeweils eine eigene Clique. Das wäre aber nur möglich, wenn es höchstens $\Theta(k)$ Kanten im Graphen gibt, also $p = \mathcal{O}\left(\frac{1}{n^2}\right)$. □

Dies zeigt, dass das Zufallsgraph-Modell von Erdős und Rényi keine guten Einblicke in den Kernelisierungs-Prozess des Clique-Cover-Problems geben kann, da mit hoher Wahrscheinlichkeit man nur „Nein"-Instanzen erhält. Man könnte nun versuchen das Modell auf „Ja"-Instanzen zu beschränken. Jedoch stellt dabei die Kontrolle der Abhängigkeiten zwischen den Kanten eine große Schwierigkeit dar. Deshalb gehen wir einen anderen Weg und betrachten das alternative Zufallsgraph-Modell der Zufalls-Schnittgraphen, welches auf einfacherem Wege nur „Ja"-Instanzen liefert.

3.4 Zufalls-Schnittgraphen

In Gramm et al. [54] werden einfache Reduktionsregeln (erstmals eingeführt in Gyárfás [58]) für die Kernelisierung des Clique-Cover-Problems untersucht. Experimentelle Ergebnisse zeigen, dass diese Regeln sehr effektiv sind. Jedoch waren die Testfälle (Erdős-Rényi-Zufalls-Graphen mit Kantenwahrscheinlichkeiten von $p = 0,15$, $0,1$ bzw. $\log(n)/n$) so klein (mit Knotenanzahlen von $n = 85$, 150 bzw. 1000), dass dafür noch die meisten Instanzen in dieser Untersuchung „Ja"-Instanzen waren. Dies ist aber, wie in Lemma 3.3.1 gezeigt, mit steigender Knotenanzahl nicht mehr der Fall.

Das motiviert die Betrachtung eines Zufallsgraph-Modells, bei welchem wir wissen, dass es nur „Ja"-Instanzen erzeugt. Wir untersuchen das bekannte Zufallsgraph-Modell der Zufalls-Schnittgraphen, welches eine Überdeckung der dabei erzeugten Graphen mit maximal k Cliquen sichert, wobei k eine Funktion $k(n)\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ ist. Dazu nutzen wir die in Abschnitt 2.7.3 gegebene Definition 2.7.2. Dort sind auch einige Eigenschaften solcher Zufallsgraphen angegeben.

Die Idee, jedem Knoten einen Binär-Vektor zuzuweisen, welcher die Zugehörigkeit des Knotens zu den überdeckenden Cliquen codiert, erschien zuerst im Beweis der beschränkten Kern-Größe durch Gyárfás [58].

Mit ähnlichen Argumenten wie in Gyárfás [58] können wir eine maximale Kerngröße von $2^k - 1$ (also nicht nur 2^k) zeigen, da auch der Nullvektor eliminiert wird (siehe Satz 3.5.1 unten). Als erstes stellen wir nun fest, dass nur „Ja"-Instanzen eine positive Auftrittswahrscheinlichkeit in unserem Modell der Zufalls-Schnittgraphen besitzen.

Satz 3.4.1. *Seien n und k natürliche Zahlen mit $k \leq n$ sowie $0 < p < 1$. Weiterhin sei $CC_k(n, p)$ der Wahrscheinlichkeitsraum der Zufalls-Schnittgraphen. Dann besitzt ein Graph darin genau dann eine positive Auftritts-Wahrscheinlichkeit, wenn er eine Clique-Überdeckung der Größe höchstens k besitzt.*

Beweis. Man beachte, dass mit $p \in (0, 1)$ auch $q \in (0, 1)$ ist.

Zuerst betrachten wir die Rückrichtung der Äquivalenzaussage: Besitzt ein Graph eine Überdeckung durch die k Cliquen C_1 bis C_k (wobei manche möglicherweise leer sein mögen), so können, wie folgt, den Knoten Binärvektoren zugeordnet werden: Ist der Knoten v Teil der Clique C_i, so setze die i-te seines Vektors c_v auf 1, andernfalls auf 0. Eine solche Verteilung von Vektoren besitzt eine echt positive Wahrscheinlichkeit.

Umgekehrt besitzt aber jeder Graph mit positiver Wahrscheinlichkeit eine solche Zuordnung von Binärvektoren zu den Knoten, sodass nach Definition alle Knoten mit Eintrag 1 an der i-ten Stelle ihrer jeweiligen Vektoren eine Clique C_i bilden und all diese Cliquen nach Definition alle Kanten im Graphen überdecken. □

In Zufalls-Schnittgraphen ist – wie bei Erdős-Rényi-Graphen – für jeden Knoten der erwartete Grad gleich; sie erzeugen homogene Zufallsgraphen. Bei Zufalls-Schnittgraphen sind jedoch die Kanten nicht unabhängig voneinander. Dies würde die Eigenschaft der gezogenen Graphen, in jedem Fall eine Clique-Überdeckung der Größe höchstens k zu besitzen, zerstören. Diese Abhängigkeiten sind dabei aber sicherlich von einer anderen Struktur, als würde man Erdős-Rényi-Graphen auf die „Ja"-Instanzen einschränken. Der Vorteil von Zufalls-Schnittgraphen dabei ist, dass wir mit den den Knoten zugehörigen Vektoren ein Werkzeug besitzen, dass uns erlaubt, diese Abhängigkeiten zu untersuchen. Für die eingeschränkten Erdős-Rényi-Graphen ist dies so nicht der Fall.

3.5 Reduktionsregeln

Wir untersuchen zwei Mengen von Reduktionsregeln. Die erste Menge ist gegeben durch die folgenden drei Regeln, welche in Gramm et al. [54] eingeführt werden.

Reduktions-Regelmenge 1. Wende folgende Regeln an:

(1.1) Lösche einen Knoten, wenn er isoliert ist, oder nur mit schon überdeckten Kanten verbunden ist.
(1.2) Ist eine noch nicht überdeckte Kante nur in einer maximalen Clique C, dann markiere deren Kanten als überdeckt und verringere den Parameter k um 1.
(1.3) Von zwei Knoten mit gleicher abgeschlossener Nachbarschaft lösche einen.

Diese Regelmenge kann in in der Knotenanzahl n polynomieller Zeit zur Reduktion des Graphen angewendet werden. Nachdem keine dieser Regeln mehr anwendbar ist, hat man einen Problemkern der Größe höchstens 2^k (oder der Eingabe-Graph war eine „Nein"-Instanz). Diese Schranke an die Kerngröße sowie die Korrektheit der Regeln (d. h., die Anwendung der Regeln führt nie dazu, dass aus einer „Ja"- eine „Nein"-Instanz, oder umgekehrt, wird) wurde durch Gramm et al. [54] bewiesen.

Um diese Regelmenge zu untersuchen, ist es von Vorteil, zuerst die folgende, ähnliche Regelmenge zu betrachten, welche stärker die Eigenschaften des Modells der Zufalls-Schnittgraphen ausnutzt.

Reduktions-Regelmenge 2. Wende die folgenden Reduktionsregeln an:

(2.1) Lösche einen Knoten, wenn sein zugeordneter Vektor der Nullvektor ist.
(2.2) Von zwei Knoten, deren zugeordnete Vektoren identisch sind, lösche einen.

Diese Regelmenge gibt hinreichende Bedingungen für die Anwendbarkeit der durch Gramm et al. [54] gegebenen Regeln:

Satz 3.5.1. *Es seien n, k positive ganze Zahlen, $0 < p < 1$ und G ein gemäß der Verteilung $CC_k(n, p)$ gezogener Graph. Wenn eine der Regeln aus Reduktionsregel-Menge 2 angewendet werden kann, dann gilt dies auch für die entsprechende Regel aus Reduktionsregel-Menge 1.*

Beweis. Beide Regeln aus Reduktionsregel-Menge 2 werden einzeln betrachtet:

(1) Ein Knoten, welchem der Nullvektor zugewiesen wurde, ist isoliert und kann damit nach Regel (1.1) gelöscht werden.
(2) Wenn es zwei verschiedene Knoten v und w mit identischen zugehörigen Vektoren c_v und c_w gibt, so sind die beiden Knoten miteinander verbunden (oder beide isoliert) und besitzen die gleichen von v und w verschiedenen Nachbarn. Also kann nach Regel (1.3) (oder (1.1)) einer der beiden eliminiert werden. □

Die in Reduktionsregel-Menge 2 definierte Regelmenge ist schwächer als die „originale" Reduktionsregel-Menge 1. Damit ist die Kerngröße nach der Reduktion durch Reduktionsregel-Menge 2 eine obere Schranke an diejenige, welche man durch Anwendung von Reduktionsregel-Menge 1 erhält.

3.6 Analyse der Kernelisierung mit Reduktionsregel-Menge 2

Wir können nun unseren ersten Satz über die Eigenschaften der betrachteten Regelmenge beweisen. Er zeigt, dass in dichten Graphen (d. h. erwarteter Grad von $\Theta(n)$) die Reduktionsregel-Menge 2 mit hoher Wahrscheinlichkeit einen Problemkern exponentieller Größe in k hinterlässt, während in dünnen Graphen (erwarteter Grad von $\mathcal{O}\left(n^{1-a}\right)$ mit einer positiven Konstanten a) mit hoher Wahrscheinlichkeit die Kern-Größe durch ein Polynom in k (mit von a abhängigem Grad) nach oben beschränkt werden kann. In allen vom letzten Fall des Satzes verschiedenen Fällen sind die angegebenen Kerne eindeutig bis auf Isomorphie. In Abbildung 3.1 werden die Ergebnisse graphisch dargestellt.

Satz 3.6.1. *Sei $k > 2$ eine feste ganze Zahl und $n \to \infty$. Sei $p(n)$ eine Funktion $\mathbb{N} \to \left(0, 1 - \left(\frac{3}{4}\right)^k\right)$ mit existierendem Grenzwert $0 \leq a := \lim_{n\to\infty} -\frac{\log p}{\log n}$ sowie $CC_k(n,p)$ der Wahrscheinlichkeitsraum der Zufalls-Schnittgraphen aus Definition 2.7.2. Dann verbleibt nach Anwendung der Reduktionsregel-Menge 2 mit hoher Wahrscheinlichkeit ein Problemkern der Größe*

- $2^k - 1$ *,wenn* $a < \frac{2}{k}$,
- $\sum_{i=1}^{\ell} \binom{k}{i} = \frac{1}{\ell!}k^\ell + \Theta\left(k^{\ell-1}\right)$, *wenn* $\frac{2}{\ell+1} < a < \frac{2}{\ell}$, *für ein festes, ganzes* $1 \leq \ell < k$,
- *0, wenn* $2 < a$ *und*
- $\sum_{i=1}^{\ell-1} \binom{k}{i} + c \cdot \binom{k}{\ell} = \frac{c}{\ell!}k^\ell + \Theta\left(k^{\ell-1}\right)$ *mit einer Konstanten* $0 < c < 1$ *(die abhängig vom genaueren Verhalten von* $\frac{\log p}{\log n}$ *ist), wenn* $a = \frac{2}{\ell}$ *für eine positive ganze Zahl* $\ell \leq k$ *ist.*

Beweis. Sei a eine positive reelle Zahl und $p(n)$ eine Funktion, welche für jedes $\varepsilon > 0$ in $\mathcal{O}\left(n^{-a-\varepsilon}\right) \cap \Omega\left(n^{-a+\varepsilon}\right)$ liegt. Dies ist äquivalent zur Aussage, dass der Grenzwert $\lim_{n\to\infty} -\frac{\log p(n)}{\log n}$ existiert und gleich a ist.

Zuerst beweisen wir eine ähnlich lautende Aussage für die „Vektor-Wahrscheinlichkeit“ $q := \left(1 - (1-p)^{\frac{1}{k}}\right)^{\frac{1}{2}}$, d. h., $\log q = \frac{1}{2} \cdot \log\left(1 - (1-p)^{\frac{1}{k}}\right)$. Für diese erhalten wir mittels der Taylor-Entwicklung des Arguments im Logarithmus an der Stelle $p_0 = 0$, dass $1 - (1-p)^{\frac{1}{k}} = \frac{1}{k} \cdot p + \mathcal{O}(p^2)$ gilt. Logarithmiert man dies, gelangt man zu

$$\log q = \frac{1}{2} \cdot (\log p - \log k) + \mathcal{O}(\log(1+p)) = \frac{1}{2}\log p + \mathcal{O}(1).$$

Damit erhält man weiterhin

$$\lim_{n\to\infty} -\frac{\log q}{\log n} = \lim_{n\to\infty} -\frac{1}{2}\cdot\frac{\log p + \mathcal{O}(1)}{\log n} = \frac{1}{2}\cdot\lim_{n\to\infty} -\frac{\log p}{\log n} = \frac{1}{2}\cdot a,$$

woraus

$$q(n) \in \mathcal{O}\left(n^{-\frac{1}{2}(a-\varepsilon)}\right) \cap \Omega\left(n^{-\frac{1}{2}(a+\varepsilon)}\right) \text{ für jedes } \varepsilon > 0 \text{ folgt.}$$

Sei nun $0 < \ell \leq k$ eine ganze Zahl und v der Binärvektor, der in den ersten ℓ Komponenten den Wert 1 und anderenfalls 0 besitzt. Die Wahrscheinlichkeit für diesen Vektor beträgt $q^\ell\cdot(1-q)^{k-\ell} = \mathcal{O}\left(n^{-\frac{1}{2}(a-\varepsilon)\cdot\ell}\right)$. Mit dem Gewicht eines Vektors bezeichnen wir im Folgenden die Anzahl seiner von 0 verschiedenen Einträge. Damit hat jeder Vektor mit Gewicht ℓ natürlich die gleiche Wahrscheinlichkeit und wegen $q = \left(1-(1-p)^{\frac{1}{k}}\right)^{\frac{1}{2}} < \left(1-\left(1-\left(1-\left(\frac{3}{4}\right)^k\right)\right)^{\frac{1}{k}}\right)^{\frac{1}{2}} = \left(1-\frac{3}{4}\right)^{\frac{1}{2}} = \frac{1}{2}$, also $q < 1-q$, ist die Wahrscheinlichkeit eines Vektors höheren Gewichts höchstens so groß. Da höchstens 2^k verschiedene solche Vektoren existieren, ist die Wahrscheinlichkeit für das Auftreten eines solchen Vektors mit Gewicht mindestens ℓ in $\mathcal{O}\left(n^{-\frac{1}{2}(a-\varepsilon)\cdot\ell}\right)$, wobei die $\mathcal{O}$-Konstante nur von k abhängt.

Ist $\frac{1}{2}a\cdot\ell > 1$, dann existiert ein $\varepsilon > 0$, sodass auch $\frac{1}{2}(a-\varepsilon)\cdot\ell > 1$ ist. Dann ist die Wahrscheinlichkeit eines solchen Vektors also in $o\left(\frac{1}{n}\right) = \frac{o(1)}{n}$. Demzufolge hat mit Wahrscheinlichkeit $1-\frac{o(1)}{n}$ ein zufällig gezogener Vektor ein Gewicht von höchstens $\ell-1$. Darüber hinaus haben damit mit Wahrscheinlichkeit $P := \left(1-\frac{o(1)}{n}\right)^n$ alle n für die Knoten im Graphen gezogenen Vektoren ein Gewicht von höchstens $\ell-1$. Mit gegen unendlich gehender Knotenanzahl n strebt P gegen 1. Demnach sind mit hoher Wahrscheinlichkeit nur Vektoren mit höchstens $\ell-1$ von 0 verschiedenen Einträgen im betrachteten Graph vorhanden. Nach dem Löschen von Duplikaten mit Reduktionsregel (2.2) verbleibt nur maximal ein Vektor jeden Typs.

Ist andererseits $\frac{1}{2}a\cdot\ell < 1$, dann existiert auch hier ein $\varepsilon > 0$ mit $\frac{1}{2}(a+\varepsilon)\cdot\ell < 1$. Damit ist die Wahrscheinlichkeit eines solchen Vektors v mit Gewicht ℓ in $\omega\left(\frac{1}{n}\right) = \frac{\omega(1)}{n}$. Mit Wahrscheinlichkeit $1-\frac{\omega(1)}{n}$ ist demzufolge ein zufällig gezogener Vektor verschieden von v. Darüber hinaus sind mit Wahrscheinlichkeit $P := \left(1-\frac{\omega(1)}{n}\right)^n$ alle n für den Graphen gezogenen Vektoren von v verschieden. Wieder mit gegen unendlich gehender Knotenanzahl strebt P zu 0, d. h., im Graphen ist mit hoher Wahrscheinlichkeit ein Knoten enthalten, dessen zugeordneter Vektor v ist.

Zusammenfassend ergibt sich also die folgende Situation: Ist $\frac{2}{\ell+1} < a < \frac{2}{\ell}$ für eine ganze Zahl $\ell < k$, dann treten mit hoher Wahrscheinlichkeit alle Vektoren mit Gewicht $\leq \ell$ und keine mit Gewicht $> \ell$ unter den n ausgewählten auf.

Ist der Wert a aber genau $\frac{2}{\ell}$, sind mit hoher Wahrscheinlichkeit alle Vektoren mit Gewicht $< \ell$ und keiner mit Gewicht $> \ell$ vertreten. Jedoch gibt es für das Vorhandensein der Vektoren vom Gewicht exakt gleich ℓ jeweils eine positive Wahrscheinlichkeit < 1, die vom genaueren Verhalten von $\frac{\log p}{\log n}$ abhängt. Nur ein Teil dieser Vektoren wird also im Kern vertreten sein. Ist schließlich $a < \frac{2}{k}$, so werden mit hoher Wahrscheinlichkeit alle 2^k Vektoren unter den n gezogenen sein, von denen nur noch der Nullvektor gelöscht wird.

Alle Duplikate der Vektoren werden bei der Kernelisierung durch die Reduktionsregel (2.2) und der Nullvektor durch (2.1) gelöscht. Da es exakt $\binom{k}{m} = \mathcal{O}(k^m)$ verschiedene Binärvektoren mit Gewicht m gibt, führt das Summieren über die relevanten Gewichte zum gewünschten Ergebnis.

Ist nicht gerade $a = \frac{2}{\ell}$ für eine ganze Zahl $1 < \ell < k$, so ist der Typ eines jeden nach dem Reduktionsprozess noch erhaltenen Vektors (welcher angibt, in welchen Cliquen der zugehörige Knoten beteiligt ist) eindeutig und bekannt. Damit ist also auch der Problemkern in diesen Fällen bis auf Isomorphie eindeutig bestimmt. □

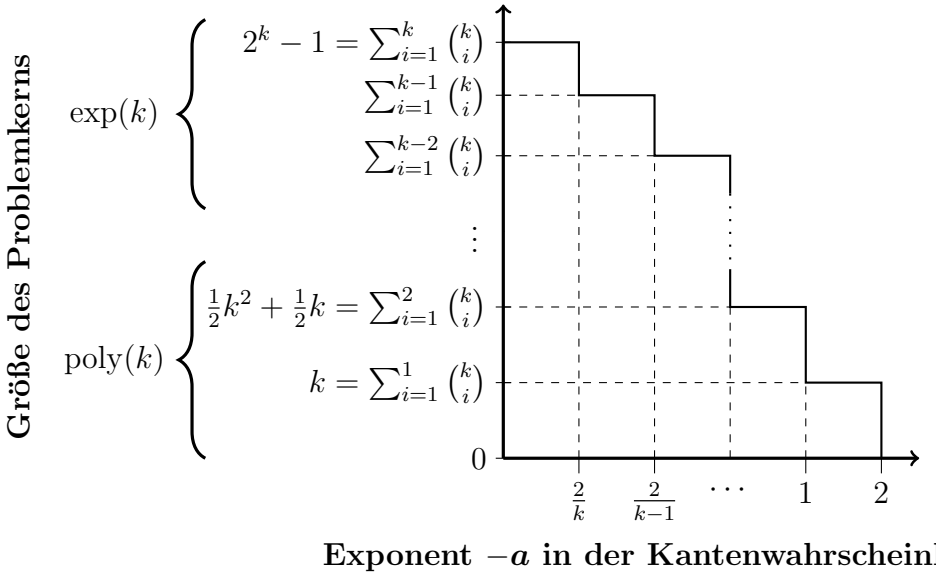

Abbildung 3.1: Illustration der resultierenden Kern-Größen als Funktion der Kantenwahrscheinlichkeit für Reduktionsregel-Menge 2, laut Satz 3.6.1: die Kern-Größe (mit hoher Wahrscheinlichkeit) als Funktion des Exponenten $a := -\lim_{n\to\infty} \frac{\log p}{\log n}$ der Kantenwahrscheinlichkeit.
Die Kerne für eine feste Kantenwahrscheinlichkeit sind dabei (mit hoher Wahrscheinlichkeit) bis auf Isomorphie eindeutig. Die gezeigten Schranken sind scharf. Nur an den Sprungstellen kann die Kerngröße (mit hoher Wahrscheinlichkeit) im angegebenen Intervall variieren.

Ist die Kantenwahrscheinlichkeit p anders als in Satz 3.6.1 nahe 1, erhalten wir das folgende, ähnliche, aber leicht verschiedene Resultat:

Korollar 3.6.2. *Sei $k > 2$ eine feste ganze Zahl und $n \to \infty$. Sei $p(n)$ eine Funktion $\mathbb{N} \to \left(1 - \left(\frac{3}{4}\right)^k, 1\right)$ mit existierendem Grenzwert $a := \lim_{n\to\infty} -\frac{\log(1-p)}{\log n}$ für eine nichtnegative Zahl a und dem in Definition 2.7.2 gegebenen Zufallsraum der Zufalls-Schnittgraphen. Dann führt die Anwendung der Reduktionsregel-Menge 2 mit hoher Wahrscheinlichkeit zu einem Problemkern der Größe*

- $2^k - 1$, *wenn $a < 1$ ist,*
- $\sum_{i=0}^{\ell} \binom{k}{i}$, *wenn $\frac{k}{\ell+1} < a < \frac{k}{\ell}$, wenn für eine feste ganze Zahl $1 \leq \ell < k$ ist,*
- *1, wenn $k < a$ ist bzw.*
- $\sum_{i=0}^{\ell-1} \binom{k}{i} + c \cdot \binom{k}{\ell}$, *wenn $a = \frac{k}{\ell}$ für eine positive ganze Zahl $\ell \leq k$ ist. Dabei ist $0 < c < 1$ eine Konstante, die vom genaueren Verhalten von $\frac{\log(1-p)}{\log n}$ abhängt.*

Beweis. Wir nutzen den Beweis von Satz 3.6.1, tauschen aber die Rollen der Einträge 0 und 1 in den Vektoren. Also muss mit $\tilde{q} := 1-q$ statt q im obigen Beweis gearbeitet werden. Ist nun $p = 1 - \mathcal{O}\left(n^{-a}\right)$, so folgt

$$q = \left(1 - (1-p)^{\frac{1}{k}}\right)^{\frac{1}{2}} = \left(1 - \mathcal{O}\left(n^{-\frac{a}{k}}\right)\right)^{\frac{1}{2}} = 1 - \mathcal{O}\left(n^{-\frac{a}{k}}\right).$$

Also ist $\tilde{q}(n) = 1 - q(n) = n^{-\frac{a}{k}}$. Dies führt mit einem analogen Vorgehen wie im Beweis von Satz 3.6.1 zu $q(n) \in \mathcal{O}\left(n^{-\frac{1}{k}(a-\varepsilon)}\right) \cap \Omega\left(n^{-\frac{1}{k}(a+\varepsilon)}\right)$ für jedes $\varepsilon > 0$. Nun können alle Berechnungen, welche Gewichte für die Vektoren vorkommen können, in genau der gleichen Weise durchgeführt werden.

Damit erhalten wir: Ist $\frac{k}{\ell+1} < a < \frac{k}{\ell}$ für eine nichtnegative ganze Zahl $\ell < k$, dann sind mit hoher Wahrscheinlichkeit alle Vektoren mit Gewicht $\geq k - \ell$ und keiner mit kleinerem Gewicht unter den n gezogenen Vektoren vertreten. Unter Benutzung von $\binom{k}{k-\ell} = \binom{k}{\ell}$ führt dies zum oben angegebenen Ergebnis. □

Diese Resultate, die jeweils „mit hoher Wahrscheinlichkeit“ gelten, übersetzen sich auch in Schranken im Average-Case.

Korollar 3.6.3. *Es seien n die Knotenanzahl, k der Parameter, p die Kanten-Wahrscheinlichkeit und $CC_k(p,n)$ der Wahrscheinlichkeitsraum der Zufalls-Schnittgraphen. Die erwartete Kerngröße, die man durch Anwendung der Reduktionsregel-Menge 2 erhält, ist dann*

- *exponentiell in k, wenn $p(n)$ konstant ist, und*
- *höchstens polynomiell in k, wenn $p(n)$ oder $1 - p(n)$ in $\mathcal{O}\left(n^{-a}\right)$ liegt.*

Beweis. Ist $p(n)$ eine Konstante, so liegt mit Wahrscheinlichkeit von $1 - o(1)$ ein Problemkern der Größe $2^k - 1$ und nur mit Wahrscheinlichkeit $o(1)$ ein kleinerer vor. Also geht mit $n \to \infty$ die erwartete Größe des Kernels gegen $2^k - 1$.

Geht $p(n)$ dagegen mindestens polynomiell gegen 0, so zeigt der Beweis von Satz 3.6.1, dass nur mit exponentiell fallender Wahrscheinlichkeit mindestens ein Vektor mit Gewicht $> \frac{2k}{a}$ im Graphen vorhanden ist. Damit (da ihre Anzahl durch ein Polynom in n beschränkt ist) fällt deren erwarteter Beitrag zur Kern-Größe auch auf 0. Analoge Ergebnisse folgen in gleicher Weise, wenn $p(n)$ mindestens polynomiell gegen 1 geht. □

3.7 Analyse der Kernelisierung mit Reduktionsregel-Menge 1

Da die Kerngröße nach Anwendung der Reduktionsregel-Menge 2 eine obere Schranke für die nach Anwendung von Reduktionsregel-Menge 1 ist, übertragen sich alle oberen Schranken aus dem vorherigen Abschnitt für Reduktionsregel-Menge 2 direkt und sind auch obere Schranken für die Kerngröße nach Anwendung von Reduktionsregel-Menge 1 durch Gramm et al. [54]. Darüber hinaus zeigen wir in diesem Abschnitt, dass die Reduktionsregel-Menge 1 den Kern weiter reduziert:

Satz 3.7.1. *Sei n eine positive ganze Zahl mit $n \to \infty$ und $k(n)$ eine Funktion $\mathbb{N} \to \mathbb{N}$. Weiterhin sei $p(n)$ eine Funktion $\mathbb{N} \to (0,1)$ mit $p < 1 - r^k$ für eine Konstante $0 < r < 1$ und schließlich $CC_k(n,p)$ der Wahrscheinlichkeitsraum der Zufalls-Schnittgraphen nach Definition 2.7.2. Dann existiert eine Konstante $c(r) > 0$, sodass die durch Gramm et al. [54] gegebene Reduktionsregel-Menge 1 mit hoher Wahrscheinlichkeit einen Problem-Kern der Größe 0 erzeugt, wenn $k(n) < c(r) \cdot \log n$ ist.*

Beweis. Zuerst bemerke man, dass wegen $p < 1 - r^k$ und der Definition des Zufallsgraph-Modells auch die Ungleichung $q < \sqrt{1-r} =: \hat{r}$ mit $0 < \hat{r} < 1$ gilt. Dies wird im Folgenden genutzt werden. Der weitere Beweis wird nach der Größe von q verschiedene Fälle unterscheiden.

Wesentliches Werkzeug in diesem Beweis werden die Vektoren vom Gewicht 1 sein. Diese charakterisieren jeweils eine eindeutige Clique. Ist also jeder solche Vektor im Graphen vorhanden, kann durch diese jede Clique eindeutig identifiziert werden (siehe Fall 2). Andernfalls (in Fall 1) zeigen wir, dass dann auch kein anderer Vektor mit höherem Gewicht bei der Konstruktion des Graphen gezogen wurde. Dann kann der Graph allein durch Anwendung von Reduktionsregel-Menge 2 vollständig reduziert werden. Man bemerke, dass wegen Satz 3.5.1 beide Regelmengen angewendet werden können.

Fall 1: $q = \mathcal{O}\left(\frac{\log k}{n}\right)$**:** Dann hat mit hoher Wahrscheinlichkeit kein Vektor ein Gewicht größer als 1. Um dies zu zeigen, bemerkt man, dass die Wahrscheinlichkeit, dass jeder bei der Konstruktion des Graphen gezogene Vektor ein Gewicht ≤ 1 besitzt, den Wert $P := \left((1-q)^k + k \cdot q \cdot (1-q)^{k-1}\right)^n$ hat. Weiterhin ist $P = \left((1-q)^k \cdot \left(1 + k \cdot \frac{q}{1-q}\right)\right)^n$. Wegen $(1-q)^k > 1 - qk$ und $1 + k \cdot \frac{q}{1-q} > 1 + kq$ ist darüber hinaus $P > ((1-qk) \cdot (1+qk))^n = (1-q^2k^2)^n$. Und wegen $k = \mathcal{O}(\log n)$ können wir P weiter abschätzen zu $\left(1 - \mathcal{O}\left(\frac{\log^2 n \cdot \log\log n}{n^2}\right)\right)^n \subseteq \left(1 - \frac{o(1)}{n}\right)^n$, was gegen 1 geht und damit mit hoher Wahrscheinlichkeit alle gezogenen Vektoren ein Gewicht von höchstens 1 besitzen.

Nach dem Löschen doppelter Vektoren mit Regel (2.2) gilt für jede der k Komponenten, dass es nur noch höchstens einen Vektor gibt, der dort einen von 0 verschiedenen Eintrag besitzt. Damit sind also nur noch isolierte Knoten im reduzierten Graphen vorhanden, welche nun durch Regel (1.1) entfernt werden. Demzufolge wird der Graph komplett reduziert, was den ersten Fall abschließt.

Fall 2: $q = \omega\left(\frac{\log k}{n}\right)$**:** Tritt der erste Fall nicht ein, so werden wir im Folgenden zeigen, dass mit hoher Wahrscheinlichkeit nun jeder Vektor mit Gewicht 1 im Graphen vorkommt. Genauer heißt dies, dass alle Vektoren v_t mit Eintrag 1 genau in der t-ten Komponente (und allen anderen Einträgen 0) für $1 \leq t \leq k$ mit hoher Wahrscheinlichkeit im Graphen vorhanden sind. Nach dem Löschen von Duplikaten gibt es dann genau einen solchen Vektor für jedes t.

Sei c_t der Knoten, der dem Vektor v_t zugehörig ist, und sei C_t die Menge der Knoten, deren zugehörige Vektoren einen 1-Eintrag an der t-ten Komponente besitzen. Nach Definition ist dies eine Clique und es ist $c_t \in C_t$. Da in v_t keine weiteren von 0 verschiedenen Einträge vorhanden sind, besitzt c_t keine anderen Nachbarn als diejenigen in C_t. Damit ist C_t eine maximale Clique und für jede Kante von c_t ist dies die einzige maximale Clique, zu der sie gehört.

Also wird die Reduktionsregel (1.2) diese Clique auswählen, damit alle ihre Kanten überdecken (und den Parameter um 1 reduzieren), was äquivalent zum Entfernen aller 1-Einträge in den t-ten Komponenten der Vektoren ist. Alle Kanten zwischen Knoten in C_t, die auch Teil einer anderen Clique C_s mit $s \neq t$ sind, werden bei diesem Prozess aber nicht gelöscht, da die ihren inzidierenden Knoten zugehörigen Vektoren noch jeweils einen weiteren von 0 verschiedenen Eintrag in der s-ten Komponente besitzen. Also können sie weiterhin zur Clique C_s beitragen.

Dieses Prozedere kann für alle Komponenten $1 \leq t \leq k$ wiederholt werden. Damit hinterlässt Reduktionsregel (1.2) nur Nullvektoren und isolierte Knoten, welche mit der Regel (1.1) nun auch gelöscht werden und also der Graph schließlich vollständig zu einem leeren Problem-Kern reduziert wurde.

Also bleibt nur noch zu zeigen, dass die Wahrscheinlichkeit P, dass wenigstens ein Vektor vom Gewicht 1 im Graphen fehlt, gegen 0 geht, wenn n gegen unendlich strebt. Sei $Q_k := q \cdot (1-q)^{k-1}$ die Wahrscheinlichkeit, dass ein konkreter Vektor mit Gewicht 1 beim Zug eines Vektors gemäß der Definition 2.7.2 gewählt wird. Dann beträgt die Wahrscheinlichkeit, dass dieser Vektor nicht Teil der n für den Graphen gezogenen ist, genau $(1-Q_k)^n$. Damit ist die Wahrscheinlichkeit P, dass mindestens einer der k Vektoren vom Gewicht 1 nicht im Graphen vorhanden ist, höchstens das k-fache dieses Werts und also $P \leq k\,(1-Q_k)^n$. Aufgrund der Ungleichung $\log(1-x) < -x$, die für alle reellen x gilt, können wir weiter abschätzen und erhalten $P \leq \exp(\log k - n \cdot Q_k)$. Damit genügt es zu zeigen, dass $Q_k = \omega\left(\frac{\log k}{n}\right)$ ist. Da Q_k eine konkave Funktion von q mit Maximum an der Stelle $q = \frac{1}{k}$ ist, bietet sich eine weitere Fallunterscheidung an, ob q größer oder kleiner als dieser Schwellwert ist. Dies erfolgt in den Fällen 2a und 2b.

Fall 2a: $q = \omega\left(\frac{\log k}{n}\right)$ und $q \leq \frac{1}{k}$: In diesem Fall ist der zweite Faktor $(1-q)^{k-1}$ von Q_k durch eine Konstante $c > 0$ nach unten beschränkt. Damit ist

$$Q_k = q \cdot (1-q)^{k-1} > cq = \omega\left(\frac{\log k}{n}\right).$$

Also wird der Graph mit hoher Wahrscheinlichkeit vollständig reduziert.

Fall 2b: $q = \omega\left(\frac{\log k}{n}\right)$ und $\frac{1}{k} < q < \hat{r}$: Der Wert von Q_k als Funktion von q ist in diesem Bereich monoton fallend. Damit ist $Q_k > \hat{r} \cdot (1-\hat{r})^k$.

Zur Vereinfachung definieren wir $c(r) := -\frac{1}{2} \cdot \frac{1}{\log(1-\hat{r})}$. Wegen $0 < \hat{r} < 1$ ist diese Konstante positiv und es ist $\log(1-\hat{r}) \cdot (c(r) \cdot \log n) = -\frac{1}{2}\log n$. Also gilt für alle $k < c(r) \cdot \log n$ die Abschätzung

$$\begin{aligned} Q_k &> \hat{r} \cdot (1-\hat{r})^k > \hat{r} \cdot \exp(1-\hat{r})^{c(r)\cdot \log n} \\ &= \hat{r} \cdot \exp(\log(1-\hat{r}) \cdot (c(r) \cdot \log n)) \\ &= \hat{r} \cdot \exp\left(-\frac{1}{2}\log n\right) = \frac{\hat{r} \cdot \sqrt{n}}{n} = \omega\left(\frac{\log k}{n}\right), \end{aligned}$$

und wieder wird der Graph auch in diesem Fall mit hoher Wahrscheinlichkeit vollständig reduziert.

Zusammenfassend konnte in jedem Fall die vollständige Reduktion gezeigt werden, was den Beweis abschließt. □

Für Kantenwahrscheinlichkeiten p, die gegen 1 gehen, lässt Satz 3.7.1 nur nichtkonstante Parameter k zu. Jedoch kann man auch für konstante k ähnliche Ergebnisse zeigen, wenn p nur höchstens „genügend langsam“ gegen 1 geht:

Korollar 3.7.2. *Unter den gleichen Voraussetzungen wie in Satz 3.7.1, aber mit konstantem k, werden gemäß $CC_k(n,p)$ gezogene Zufalls-Schnittgraphen mit der durch Gramm et al. [54] gegebenen Reduktionsregel-Menge 1 mit hoher Wahrscheinlichkeit vollständig reduziert, wenn $p < 1 - n^{-2/k}$ ist.*

Beweis. Wie in Satz 3.6.1 und der folgenden Bemerkung wissen wir, dass für konstante k und die gegebene Einschränkung an p unsere Reduktionsregel-Menge den Graphen mit hoher Wahrscheinlichkeit nur auf einen Problemkern der Größe $2^k - 1$ reduziert. Dort sind alle Vektoren mit positivem Gewicht genau einmal vorhanden, insbesondere diejenigen mit Gewicht 1. Doch dann kann der Graph vollständig durch die Anwendung von Regel (1.2) reduziert werden, wie dies im zweiten Teil des Beweises von Satz 3.7.1 gezeigt wurde. □

3.8 Zusammenfassung

Es ist bekannt, dass die parametrisierte Version des Clique-Problems in einer Worst-Case-Betrachtung W[1]-vollständig ist. Jedoch ist es in erwarteter FPT-Zeit für Erdős-Rényi-Zufallsgraphen, [43], und auf Chung-Lu-Zufallsgraphen, [45, 46], lösbar.

Hier haben wir das Average-Case-Verhalten des Clique-Cover-Problems studiert, welches in FPT liegt, aber wohl keinen subexponentiellen Problemkern im Worst-Case besitzt. Um diese Kern-Größen untersuchen zu können, muss der Fokus auf die „Ja“-Instanzen gelegt werden. Zuvor erfolgte Experimente nutzten das Modell der Erdős-Rényi-Zufallsgraphen, welches aber für eine asymptotische Analyse ungeeignet ist, da es mit hoher Wahrscheinlichkeit nur „Nein“-Instanzen erzeugt (vgl. Lemma 3.3.1). Stattdessen wurden hier Zufalls-Schnittgraphen betrachtet, welche auf natürliche Weise durch die Eigenschaft entstehen, da jede Überdeckung des Graphen mit k Cliquen eine entsprechende Zuordnung von Binärvektoren zu den Knoten besitzt. Gezeigt werden konnte für weite Bereiche der Kantenwahrscheinlichkeit p, dass die Reduktionsregel-Menge, die durch Gramm et al. [54] angegeben wurde, mit hoher Wahrscheinlichkeit den Graphen vollständig reduziert, wenn der Parameter k eine gewisse Schranke nicht übersteigt (siehe Satz 3.7.1).

4 Suchbäume

4.1 Überblick

Beschränkte Suchbaum-Algorithmen sind ein bekanntes Paradigma für das Design effizienter parametrisierter Algorithmen. Wir untersuchen die erwartete Laufzeit von drei beschränkten Suchbaum-Algorithmen auf einem verallgemeinerten Zufallsgraph-Modell sowie einem analogen Modell zufälliger Mengen-Familien. Das verwendete Zufallsgraph-Modell verallgemeinert Erdős-Rényi-Graphen und erlaubt jeder potentiellen Kante eine eigene, individuelle Wahrscheinlichkeit zu besitzen, mit der sie im gezogenen Graphen vorhanden ist. Die einzige Bedingung ist die Unabhängigkeit dieser Kantenwahrscheinlichkeiten.

Für das Vertex-Cover-Problem in Graphen mit n Knoten und einer Größe k der gesuchten Knotenüberdeckung zeigen wir eine erwartete Laufzeit von $\mathcal{O}\big(k^{\mathcal{O}(1)} \cdot n\big)$, die deutlich geringer ist als die untere Schranke in einer parameterisierten Worst-Case-Betrachtung, nach der keine Laufzeit von $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ erreicht werden kann. Für das Clique-Problem in Graphen mit n Knoten und einer Größe von k der zu findenden Clique zeigen wir eine erwartete Laufzeit von $f(k) \cdot n$ für eine Funktion $f : \mathbb{N} \to \mathbb{N}$, was analog deutlich niedriger ist als die untere Schranke einer parameterisierten Worst-Case-Betrachtung, nach der keine Laufzeit von $n^{\mathcal{O}(k)}$ erreicht werden kann. Das W[1]-vollständige Problem Clique ist damit in der Komplexitätsklasse avg−FPT für die untersuchte Verteilung der Eingabegraphen. Gleiches gilt für das W[2]-vollständige Hitting-Set-Problem, mit einer Familie $\mathcal{F}$ von Teilmengen einer n-elementigen Grundmenge, aus der eine k-elementige Teilmenge S ausgewählt werden soll, sodass jedes Element der Familie einen nicht-leeren Schnitt mit S besitzt. Auch hier erhalten wir wieder eine erwartete avg−FPT-Laufzeit. Beschränkt man die Mächtigkeit der Mengen in $\mathcal{F}$ auf höchstens d Elemente, so wird dieses Problem auch im Worst-Case FPT. Jedoch ist die Laufzeit dann exponentiell in k, während wir eine erwartete Laufzeit polynomiell in k (mit Grad abhängig von d) erhalten.

Unsere Ergebnisse gelten auch in einer Smoothed-Betrachtung, wo beliebige Worst-Case-Instanzen zufällig verrauscht und auf diesen die erwartete Laufzeit betrachtet werden. Wir zeigen, dass bei einem zufälligen und jeweils unabhängigen „Flippen“ jeder potentiellen Kante, d. h. eine Kante wird durch eine Nicht-Kante bzw. umgekehrt ersetzt, mit konstanter Wahrscheinlichkeit $\varepsilon > 0$ alle drei NP-vollständigen Probleme in erwarteter Polynomialzeit gelöst werden können.

4.2 Einführung

Die klassische Sicht in der Komplexitätstheorie ist eine Worst-Case-Perspektive: Wie lang benötigt ein Algorithmus höchstens, um eine Instanz der Größe n zu bearbeiten? Damit erhält man eine Unterscheidung in „leichte“ Probleme in P mit

Polynomialzeit-Algorithmen und „schwierigen“ Problemen in NP, wo nur Algorithmen mit exponentieller Laufzeit bekannt sind. Mit einer *parametrisierten* Sicht, die von Downey und Fellows [36] eingeführt wurde, erhält man genauere Einblicke. Relevante Definitionen dazu wurden bereits in Abschnitt 2.2 gegeben. Für das Vertex-Cover-Problem (mit zusätzlichem Parameter k der Größe der zu findenden Knotenüberdeckung) existiert dabei ein Algorithmus, dessen Laufzeit in $f(k) \cdot n^{\mathcal{O}(1)}$ liegt. Damit ist das Problem „fixed-parameter tractable“ und also in FPT. Für das Clique-Problem ist kein solcher Algorithmus bekannt. Tatsächlich ist es W[1]-vollständig. Schließlich lässt sich das Hitting-Set-Problem nach der weithin angenommenen Vermutung, dass $\mathsf{W}[1] \neq \mathsf{W}[2]$ ist, nicht einmal in FPT-Zeit auf ein beliebiges Problem der parametrisierten Klasse W[1] reduzieren. Es ist W[2]-vollständig.

Der zweite Ansatz konzentriert sich nicht auf Worst-Case-Instanzen sondern den erwarteten Fall, die *Average-Case*-Instanz: Ist eine Wahrscheinlichkeitsverteilung auf den potentiellen Eingaben gegeben, kann man die erwartete Laufzeit eines Algorithmus berechnen. Auch hierzu können die relevanten Definitionen bereits in Abschnitt 2.3 nachgelesen werden.

Wenn die Average-Case-Komplexität eines Algorithmus bestimmt werden soll, ist die Wahrscheinlichkeitsverteilung der Eingaben von großer Bedeutung. Wir betrachten dabei für die beiden Graph-Probleme Graphen mit jeweils n Knoten. In vielen Untersuchungen wird hierfür das Zufallsgraph-Modell von Erdős und Rényi [39] verwendet, in welchem jede potentielle Kante unabhängig mit einer Wahrscheinlichkeit p im Graphen vorhanden ist, siehe Abschnitt 2.7.1. Wir werden hier im Folgenden ein allgemeineres Modell für Zufallsgraphen verwenden, welches wir in Abschnitt 2.7.4 definiert haben.

Während sich eine Worst-Case-Analyse nur auf die „schwierigen“ Instanzen konzentriert, „glättet“ die Beschränkung auf den Erwartungswert womöglich alle Schwierigkeiten. Damit ist es interessant, zwischen diese beiden Extrempositionen zu schauen: Dies tut das Konzept der *Smoothed-Komplexität*, wie wir es in Abschnitt 2.4 bereits kennengelernt haben. Dabei wird die Frage untersucht, wie schwierig Instanzen noch sind, nachdem sie ein gewisses, zufälliges „Verrauschen“ durchlaufen haben. Eine ganze Reihe an Untersuchungen, wie sich Algorithmen unter diesem Komplexitätsbegriff verhalten, findet man im angegebenen Abschnitt. Jedoch ist dem Autor kein Ergebnis aus der Literatur zur Smoothed-Komplexität für ein parametrisiertes Problem bekannt, auch nicht für die klassischen Prototypen parametrisierter Probleme Vertex-Cover, Clique und Hitting-Set.

Jeder dieser verschiedenen Ansätze ist für sich gut untersucht. In diesem Abschnitt sollen sie kombiniert werden, um noch bessere Resultate zu erhalten: Was kann man lernen, wenn man das Paradigma der beschränkten Suchbäume, die häufig beim Design von Algorithmen für parametrisierten Algorithmen verwendet werden, in einer Average-Case- oder Smoothed-Komplexitäts-Untersuchung einsetzt?

Fountoulakis et al. [43] geben eine Average-Case-Untersuchung des parametrisierten Clique-Problems an, bei dem der Parameter k die Größe der zu findenden Clique ist. Dabei verwenden sie als Eingabeverteilung das Zufallsgraphen-Modell von Erdős-Rényi und konnten zeigen, dass Clique (mit diesem Parameter) in $\mathsf{avg}-\mathsf{FPT}$ ist. Jedoch lässt sich dieses Ergebnis nicht zu einem Ergebnis der Smoothed-Komplexität verallgemeinern, da explizit die Identität aller Kantenwahrscheinlichkeiten genutzt wird. Auch wurden dort nur Kantenwahrscheinlichkeiten untersucht, die eine Funktion in der Anzahl der Knoten n sind, nicht aber vom Parameter k abhängig sein dürfen.

Beide Beschränkungen werden in dieser Untersuchung abgeschwächt. Dies geschieht auf Kosten des Bereichs der zulässigen Kantenwahrscheinlichkeiten, sodass nicht mehr der gesamte Bereich von möglichen Wahrscheinlichkeitsverteilungen durch die Sätze abgedeckt werden kann. Für das allgemeinere Mengen-Problem Hitting-Set-Problem verallgemeinern wir das hier verwendete Zufallsgraph-Modell weiter, sodass es auch für dieses Problem anwendbar wird. Darüber hinaus wird hier das klassische Paradigma der beschränkten Suchbäume untersucht. Dass dieses generell gute Ergebnisse im erwarteten Fall liefert, wird dabei nicht nur am Clique- sondern auch dem Vertex-Cover- und dem Hitting-Set-Problem gezeigt.

Unsere Ergebnisse In unseren Untersuchungen betrachten wir das sehr allgemeine Zufallsgraph-Modell von [73], in welchem jede potentielle Kante (i, j) unabhängig von den anderen jeweils mit einer individuellen Wahrscheinlichkeit $p_{i,j}$ im Graphen erscheint. Um einen Kollaps zu einer Worst-Case-Instanz zu vermeiden, werden wir in den Sätzen annehmen, dass alle Kantenwahrscheinlichkeiten (p, q)-beschränkt sind, d. h., jeweils $p \leq p_{i,j} \leq q$ gilt. Im Fall des Hitting-Set-Problems nehmen wir an, dass jede Teilmenge $S \subset G$ der Grundmenge G eine individuelle Auftrittswahrscheinlichkeit besitzt. Dabei können z. B. auch alle solchen Teilmengen mit weniger als d Elementen die Wahrscheinlichkeit 0 zugewiesen bekommen, womit dann das verwandte d-Hitting-Set-Problem betrachtet wird. Genauere Details der beiden Modelle wurden bereits in Abschnitt 2.7.4 beschrieben.

Die untersuchten Algorithmen hängen nicht von den speziellen Eigenschaften des zugrunde liegenden Wahrscheinlichkeitsraums (wie dies in [43] der Fall ist) ab, sondern nutzen das Paradigma der *beschränkten Suchbaum-Algorithmen*, welches eine Standard-Technik beim Design von Algorithmen für parametrisierte Probleme ist. In Abschnitt 4.3.2 beschreiben wir die allgemeine Struktur eines beschränkten Suchbaum-Algorithmus im Zusammenhang mit der Suche nach einer Teilmenge der Knoten eines Graphen mit einer bestimmten Eigenschaft. (Ein analoges Vorgehen findet dann auch bezüglich Hitting-Set statt, auch wenn es sich dort formal nicht um einen Graphen handelt.) In den zentralen Abschnitten 4.4, 4.5 und 4.6 untersuchen wir die auf die jeweiligen Probleme angepassten Versionen des beschränkten Suchbaum-Algorithmus, welche (wenn existent) eine Knotenüberdeckung aus maximal k Knoten (Algorithmus 4.3), eine Clique der Größe mindestens k (Algorithmus 4.4) bzw. ein Hitting-Set aus höchstens k Elementen (Algorithmus 4.5) finden.

Tabelle 4.1 gibt dabei einen Überblick über bisher bekannte und unsere neuen Ergebnisse zu verschiedenen Komplexitäts-Eigenschaften für Vertex-Cover und Clique. Wir zeigen für beide Probleme, dass sie in avg−FPT liegen (Sätze 4.4.4 und 4.5.4), d. h., es gibt für sie Algorithmen mit erwarteter FPT-Laufzeit. Da das Vertex-Cover-Problem selbst schon in einer Worst-Case-Betrachtung in FPT liegt, ist dies hier nicht überraschend. Jedoch ist die Abhängigkeit der Laufzeit vom Parameter k nicht nur subexponentiell, sondern für weite Bereiche auch polynomiell (Satz 4.4.4). Unter Annahme der Exponentialzeit-Hypothese [63] ist dies für den Worst-Case unmöglich zu erreichen. Für das W[1]-vollständige parametrisierte Clique-Problem zeigen wir, dass die erwartete Laufzeit des angegebenen Algorithmus nicht nur in avg−FPT liegt, sondern für kleine Werte des Parameters k sogar linear in n ist (Satz 4.5.4). Weiterhin untersuchen wir ein Smoothed-Modell, in welchem, startend von einer Worst-Case-Instanz, jede potentielle Kante darin unabhängig mit Wahrscheinlichkeit ε „geflippt“ wird. (Eine formale Definition ist in Abschnitt 4.3.1 zu finden.) Für eine konstante Verrauschungs-Wahrscheinlichkeit $\varepsilon > 0$ ergibt sich dann eine polynomielle erwartete Laufzeit für beide hier betrachteten NP-vollständigen Probleme (Sätze 4.4.8 und 4.5.6).

	Vertex-Cover		**Clique**	
Klassische Komplexität	NP-vollständig	[65]	NP-vollständig	[65]
Parametrisierte Komplexität	FPT	[36]	W[1]-vollständig	[36]
Avg.-case Param. Komp.	avg−FPT	Satz 4.4.4	avg−FPT	Satz 4.5.4
Worst-Case Laufzeit	$\mathcal{O}(1.1889^n)$	[90]	$\mathcal{O}(1.1889^n)$	[90]
Avg.-Case Laufzeit	$n^{\mathcal{O}(\log n)}$	[7]	$n^{\mathcal{O}(\log n)}$	[7]
Param. Laufzeit	$\mathcal{O}(1.2738^k + kn)$	[26]	$\mathcal{O}(k^2 \cdot n^k)$	(nach Def.)
Param. Untere Schranke	$2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$	[23]	$n^{\mathcal{O}(k)}$	[25]
Avg.-Case Param. Laufzeit	$\text{poly}(k) \cdot n$	Satz 4.4.4	$f(k) \cdot n$	Satz 4.5.4
Smoothed-Komplexität	$\text{poly}(n)$	Satz 4.4.8	$\text{poly}(n)$	Satz 4.5.6

Tabelle 4.1: Überblick zu bekannten und neuen Komplexitäts- und Laufzeitergebnissen für Vertex-Cover und Clique. Die unteren Schranken gelten unter Annahme der Exponentialzeit-Hypothese [63]. In allen Fällen ist n die Anzahl der Knoten und k die Größe der Knotenüberdeckung bzw. Clique. Die Average-Case-Ergebnisse gelten für die Eingabeverteilung $G(n, p)$ (klassische Komplexität) bzw. $RG(n, p, q)$ (bei der parametrisierten Komplexität; das Zufallsgraph-Modell wird in Definition 2.7.4 eingeführt). Unsere Average-Case-Resultate gelten nur für die Werte der Kantenwahrscheinlichkeiten p, ε und q, welche genügend weit von 0 und 1 entfernt sind, sowie gegebenenfalls genügend kleinen Werten des Parameters k. Alle Voraussetzungen sind in den jeweils angegebenen Sätzen aufgelistet.

Weiterhin gibt Tabelle 4.2 einen Überblick über bekannte und hier neu gefundene Ergebnisse zu verschiedenen Komplexitäts-Ergebnissen von (d)-Hitting-Set. Ähnlich wie für Vertex-Cover und Clique zeigen wir, dass die beiden Probleme in avg−FPT liegen (Sätze 4.6.8 und 4.6.10), d. h., es gibt für sie Algorithmen mit erwarteter FPT-Laufzeit. Während dies für das schon im Worst-Case in FPT liegende d-Hitting-Set-Problem nicht weiter verwunderlich ist, konnten wir dort aber für konstante Wahrscheinlichkeiten p statt einer exponentiellen eine polynomielle Abhängigkeit der Laufzeit vom Parameter k nachweisen (Korollar 4.6.9). Ein ähnliches Ergebnis erhalten wir darüber hinaus auch für das im Worst-Case W[2]-vollständige Hitting-Set-Problem (Korollar 4.6.11). Diese übertragen sich dann auch auf die Untersuchung in einem Smoothed-Modell (Satz 4.6.12 und Korollar 4.6.14), was zeigt, dass sowohl das spezielle als auch das allgemeine Hitting-Set-Problem bei gestörten Daten im Erwartungswert recht schnell gelöst werden kann.

	d-Hitting-Set		**Hitting-Set**	
Klassische Komplexität	NP-vollständig	[65]	NP-vollständig	[65]
Parametrisierte Komplexität	FPT	[36]	W[2]-vollständig	[36]
Avg.-case Param. Komp.	avg−FPT	Satz 4.6.8	avg−FPT	Satz 4.6.10
Param. Laufzeit	$\mathcal{O}(d)^k + \mathcal{O}(\lvert\mathcal{F}\rvert)$	[41]	$\mathcal{O}(n^k \cdot \lvert\mathcal{F}\rvert)$	(nach Def.)
Avg.-Case Param. Laufzeit	$\mathcal{O}\left(k^{d+c} \cdot \lvert\mathcal{F}\rvert\right)$	Kor. 4.6.9	$\mathcal{O}\left(k^c \cdot \lvert\mathcal{F}\rvert\right)$	Kor. 4.6.11
Param. Smoothed-Komp.	$\mathcal{O}\left(k^{d+c} \cdot n^d\right)$	Satz 4.6.12	$k^c \cdot \mathcal{O}\left(n^c + \lvert\mathcal{F}\rvert\right)$	Kor. 4.6.14

Tabelle 4.2: Überblick zu bekannten und neuen Komplexitäts- und Laufzeitergebnissen für (d)-Hitting-Set. In allen Fällen ist n die Größe der Grundmenge, k die Größe des Hitting-Sets, d die Größe der einzelnen Mengen und $|\mathcal{F}|$ die der Mengenfamilie sowie $c > 0$ eine geeignet gewählte (von p bzw. ε abhängige) Konstante. Die Average-Case-Ergebnisse gelten für die Eingabeverteilungen $RS_d(n,p)$ bzw. $RS(n,p)$. Unsere Average-Case-Resultate gelten nur für die Werte der Kantenwahrscheinlichkeiten p und ε, welche genügend weit von 0 entfernt sind, sowie genügend kleinen Werten der Parameter d und k. Alle Voraussetzungen sind in den jeweils angegebenen Sätzen aufgelistet.

4.3 Voraussetzungen

In diesem Abschnitt seien $n \in \mathbb{N}$, $k\colon \mathbb{N} \to \mathbb{N}$ mit $0 \leq k(n) \leq n$ und, wenn nicht anders angegeben, $\varepsilon > 0$ eine reelle Konstante. Für eine Funktion $f : \mathbb{N} \to \mathbb{R}_{>0}$ seien mit $o(f), \mathcal{O}(f), \omega(f)$ und $\Omega(f)$ die entsprechenden Landau-Symbole bezeichnet.

Weiterhin nutzen wir hierbei das Modell „Allgemeiner Zufallsgraphen“, wie wir es in Definition 2.7.4 in Abschnitt 2.7.4 eingeführt haben. Dies wird uns später die Freiheit geben, nicht nur Average-Case- sondern auch Smoothed-Aussagen zu formulieren.

4.3.1 Smoothed-Komplexität

Um bessere Einsichten zu erhalten, wie sich Probleme zwischen Worst- und Average-Case verhalten, führten Spielman und Teng [95] das Konzept der Smoothed-Komplexität ein. In Abschnitt 2.4 haben wir bereits in Definition 2.4.1 den Begriff der Smoothed-Komplexität im Kontext eines graphentheoretischen Algorithmus angegeben, den wir im Folgenden hier verwenden werden.

In deren Arbeit argumentieren Spielman und Teng [94], dass die Störung aller Kanten eines Graphen wichtige Eigenschaften dieses Graphens zerstören kann. So kann sich z. B. die Größe einer Maximum-Clique in diesem Graphen deutlich verändern. Dennoch glauben wir, dass eine Untersuchung der Smoothed-Komplexität in einem parametrisierten Umfeld von eigenem Interesse ist.

Auf analoge Weise erhält man auch die Definitionen der Smoothed-Komplexität für Probleme, deren Eingaben gemäß unseres Zufalls-Mengenfamilien-Modells verteilt sein können, sowie in beiden Fällen für die entsprechenden parametrisierten Varianten.

4.3.2 Das Paradigma beschränkter Suchbäume

Eine weitverbreitete Strategie beim Lösen parametrisierter Probleme ist die Technik der beschränkten Suchbäume. Dabei wird das Problem als Suchproblem in einem bestimmten Baum reformuliert, welcher dann per Tiefensuche oder anderer Methoden untersucht werden kann. Ist der Parameter des ursprünglichen Problems fest gewählt und besitzt der Suchbaum nur eine von diesem Parameter abhängige, beschränkte Verzweigung sowie Tiefe, erhält man damit einen FPT-Algorithmus. (Jedoch ist dies keine notwendige Voraussetzung, wie z. B. die Autoren in [96] zeigen. Dort ist die hier beschriebene Beschränkung durch den Parameter an die Anzahl der Verzweigungen in einem Knoten des Suchbaums nicht gegeben. Sie erhalten aber dennoch eine FPT-Laufzeit, da in jedem Schritt der Wert des Parameters nun um einen vom Verzweigungsgrad abhängigen Wert reduziert wird.)

Im Folgenden werden wir einen speziellen Typ solcher beschränkter Suchbaum-Algorithmen für Graphen-Probleme betrachten, in welchen die „Kind-Knoten“ im Suchbaum (bzw. äquivalent, die zu lösenden Teilaufgaben des Ausgangsproblems) zu Instanzen des gleichen Problems korrespondieren, die als induzierter Teilgraph einer bestimmten Knoten-Teilmenge des Ausgangsggraphen und eventuell reduziertem Parameter entstehen. Damit werden unsere Algorithmen jeweils prinzipiell die Form des Algorithmus 4.1, der diese allgemein darstellt, besitzen.

Algorithmus 4.1 : Allgemeiner Aufbau eines beschränkten Suchbaum-Algorithmus

Input : Graph $G = (V, E)$ und Parameter k
Output : Existiert eine Lösungsmenge $S \subset V$ mit $|S| = k$? (Wenn „ja“, gib eine solche Lösung zurück.)

1 Wähle einen Knoten $v \in V$.
`// Nimm zuerst an, dass` v `Teil der Lösung ist. Löse das ,,Restproblem'' rekursiv.`
2 Bestimme eine geeignete, problemspezifische Knoten-Teilmenge $V' \subset V \setminus \{v\}$ und löse das Problem rekursiv auf dem induzierten Teilgraphen $G[V']$ von G mit zugehörigem Parameter $k' < k$.
3 **if** *Wurde eine Lösung gefunden?* **then**
4 | **return** *„ja“ und die Lösungsmenge vereinigt mit* $\{v\}$ *selbst.*
5 **else** `// Wurde keine Lösung gefunden, in der` v `enthalten ist, suche nach einer ohne diesen Knoten.`
6 | Lösche v (und problemspezifisch eventuell zusätzliche Knoten) und löse das Problem rekursiv auf diesem induzierten Teilgraphen.
7 | **if** *Wurde eine Lösung gefunden?* **then return** *„ja“ und die entsprechende Lösung.*
8 **return** *„nein“.* `// Alle Rekursionsaufrufe gaben nur` ***Nein***`-Antworten zurück. Also existiert keine Lösung.`

Es ist allgemein bekannt, dass bei einer solchen Suche, wie wir sie in Algorithmus 4.1 beschrieben haben, nicht notwendigerweise alle Zweige des Suchbaums ausgewertet werden müssen:

Findet man eine Lösung im ersten Teilbaum, kann man die weiteren außen vor lassen, wie dies etwa in [77] beschrieben wird.

Und formulieren wir zusätzlich Schritt 6 von Algorithmus 4.1 so um, wie im äquivalenten Algorithmus 4.2 angegeben, können wir auch sicherstellen, dass in jedem Rekursionsaufruf die Anzahl der Knoten im betrachteten Graphen um mindestens 1 fällt. Dies wird bei der Analyse solcher Algorithmen helfen.

Algorithmus 4.2 : Allgemeiner Aufbau mit Reduktion der Knotenzahl in jedem Rekursionsaufruf

`// Algorithmus 4.1 mit Ersetzung des Schrittes 6 durch`
6′ Ist v nicht Teil der Lösung, lösche diesen (und problemspezifisch eventuell weitere).
6″ Ist nun die Knotenmenge V leer, gib eine „nein“-Antwort zurück, sonst gehe zu Schritt 1.

Wir möchten die erwartete Laufzeit solcher Algorithmen in Abhängigkeit von Verteilungen $RG(n, p)$ der Eingabe-Graphen untersuchen. Diese sind also Zufallsgraphen mit unabhängig gezogenen Kanten. Unsere Algorithmen untersuchen jeweils einen Knoten nach dem anderen und decken so nur sukzessive den Graphen auf. Nachdem ein Knoten betrachtet wurde, werden er und alle seine potentiellen Kanten gelöscht. Dadurch befinden sich in den induzierten Teilgraphen, die in den Rekursionsaufrufen betrachtet werden, nur bisher nicht untersuchte Knoten. Also können diese Teilgraphen genauso als Zufallsgraphen mit gleichen Parametern der zugehörigen Verteilungen betrachtet werden.

Demzufolge werden wir im Folgenden annehmen, dass diese Teilgraphen, die die Eingaben der Rekursionsaufrufe sind, „der gleichen Verteilung“ (mit geringerer Knotenanzahl) folgen wie der Eingabe-Graph selbst: Ihre Wahrscheinlichkeits-Matrizen (welche man aus der des Eingabegraphen durch Löschen der zu den gelöschten Knoten zugehörigen Zeilen und Spalten erhält) sind p-beschränkt genau dann, wenn die Wahrscheinlichkeits-Matrix des Eingabegraphen selbst p-beschränkt ist. Mit dieser Beobachtung gestaltet sich die Untersuchung der erwarteten Laufzeit solcher Algorithmen deutlich einfacher.

Wandelt man die genannten Algorithmen 4.1 bzw. 4.2 in der Art ab, dass sie nicht mehr über Knoten, sondern Elemente der Grundmenge, und nicht mehr über Kanten, sondern Mengen, die dieses Element enthalten, sprechen, dann sind sie auch für die allgemeineren Mengenprobleme wie das Hitting-Set-Problem anwendbar. Auch hier kann nun nach der gleichen Argumentation gefolgert werden, dass sich die Wahrscheinlichkeitsverteilung auf die in den Rekursionsaufrufen betrachteten Teilstrukturen überträgt. Eine genauere Untersuchung dazu findet im Abschnitt 4.6 statt.

4.4 Das Vertex-Cover-Problem

Beim (parametrisierten) Knotenüberdeckungs- oder Vertex-Cover -Problem geht es darum, eine Knotenteilmenge (der Größe k) des Eingabe-Graphen zu finden, sodass jede Kante in diesem Graphen mit mindestens einem der Knoten aus der Teilmenge verbunden ist. Gibt k dabei die Größe der zu findenden Knotenteilmenge an, wird das Problem in einer Laufzeit von $f(k) \cdot n^{\mathcal{O}(1)}$ lösbar, was bedeutet, dass Vertex-Cover in der Komplexitätsklasse FPT liegt. Jedoch ist in einer Worst-Case-Betrachtung das Vertex-Cover-Problem NP-vollständig und es sind nur Algorithmen mit exponentieller Worst-Case-Laufzeit bekannt. Unter Annahme der Exponentialzeit-Vermutung [63] kann es auch keinen Algorithmus geben, welcher Vertex-Cover garantiert in subexponentieller Laufzeit in Abhängigkeit der Knotenzahl n des Eingabe-Graphen löst.

Ist der Knoten v Teil der ausgewählten Menge, so werden alle seine Kanten schon durch ihn überdeckt. Also brauchen diese nicht mehr durch andere Knoten überdeckt werden und wir können seine Kanten mit v selbst löschen. Letzteres ist deshalb möglich, da v ja auch keine weiteren Kanten überdecken kann. Damit ist unsere Vorgehensweise in Schritt 2 von Algorithmus 4.3, in diesem Fall den Knoten v mit seinen Kanten zu löschen, bevor das „Restproblem“ rekursiv betrachtet wird, gerechtfertigt. Der Rekursionsaufruf wird dann mit dem Parameter $k' = k - 1$ durchgeführt, da mit v ja ein Element der Lösungsmenge schon gefunden wurde.

Ist im anderen Fall der Knoten v dagegen nicht Teil der zu findenden Knotenüberdeckung, so müssen es alle seine Nachbarn sein. Denn, damit jede von v ausgehende Kante dann einen von v verschiedenen Knoten überdeckt wird, muss der zugehörige andere Endknoten dieser Kante in die Lösungsmenge aufgenommen werden. Fasst man diese Beobachtungen zusammen, kann damit der allgemeine beschränkte Suchbaum-Algorithmus nun für dieses Problem in den Algorithmus 4.3 konkretisiert werden.

Algorithmus 4.3 : Einfacher beschränkter Suchbaum-Algorithmus für das VERTEX-COVER-Problem

```
// Algorithmus 4.2, in welchem die Schritte 2 bis 6' ersetzt
   werden durch
```
2 Nimm zuerst an, dass v Teil der Lösung ist: Lösche v aus G und löse das Problem rekursiv mit Parameter $k' = k - 1$.
3 **if** *Wurde eine Lösung gefunden?* **then**
4 | **return** *„ja“ und die Lösungsmenge vereinigt mit* $\{v\}$ *selbst.*
5 **else** `// Wenn keine Lösung existiert, die` v `enthält, suche nach einer ohne diesen Knoten.`
6 | Lösche v und alle seine Nachbarn. Löse das Problem rekursiv auf diesem reduzierten Graphen und Parameter $k' = k - d(v)$, wobei $d(v)$ der Grad des Knoten v im Eingabe-Graphen ist.
7 | **if** *Wurde eine Lösung gefunden?* **then return** *„ja“ und die Lösungsmenge selbst.*
8 **return** *„nein“.*

Definition 4.4.1. Sei $T_p(n, k)$ die erwartete Laufzeit von Algorithmus 4.3 bei einer p-beschränkten Wahrscheinlichkeitsverteilung $RG(n, p)$ des Eingabe-Graphen und dem Parameter k.

Dann können wir die entstehende Rekursionsgleichung für $T_p(n, k)$ unter Annahme einiger Voraussetzungen an die relevanten Parameter auflösen und erhalten die folgenden Sätze.

Satz 4.4.2. *Sei* $G \sim RG(n, p)$ *mit* $\frac{(1+\varepsilon)k}{n} \leq p \leq 1$. *Dann ist die erwartete Laufzeit von Algorithmus 4.3 gleich* $T_p(n, k) = \mathcal{O}(k \cdot n)$.

Anzumerken ist, dass in dieser Formulierung des Satzes zuerst der Parameter k und erst danach die Kantenwahrscheinlichkeit p gewählt wird. Demzufolge schränkt der für k gewählte Wert das mögliche Intervall ein, aus dem die Kantenwahrscheinlichkeit p stammen darf, damit der Satz Anwendung finden kann. Ähnliches gilt für die folgenden Sätze und deren Korollare.

Beweis. Ist die Wahrscheinlichkeitsverteilung des Eingabegraphen p-beschränkt, so ist jede Kante unabhängig von den anderen mit einer Wahrscheinlichkeit von mindestens p im Zufallsgraphen enthalten. Damit ist der erwartete Grad μ eines Knotens mindestens $\mu \geq (n-1) \cdot p \geq (1+\varepsilon) \cdot \left(1 - \frac{1}{n}\right) k \geq (1+\tilde{\varepsilon})k$, wobei $0 < \tilde{\varepsilon} < \varepsilon$ beliebig gewählte, aber feste Konstanten und n genügend groß (in Abhängigkeit dieser Konstanten) sind.

Unter Benutzung dieser Tatsache sowie der Chernoff-Schranke für die Summe unabhängiger $\{0;1\}$-verteilter Zufallsvariablen (in unserem Kontext: Ist eine bestimmte, potentielle Kante, die den betrachteten Knoten als Endpunkt besitzt, im Zufallsgraphen enthalten, oder nicht?), wie wir sie in in Lemma 2.8.1 angegeben haben, können wir eine obere Schranke an die Wahrscheinlichkeit angeben, dass der betrachtete Knoten einen Grad von höchstens k besitzt: Setze $0 < \delta := 1 - \frac{1}{1+\tilde{\varepsilon}} < 1$. Dann ist $(1-\delta)\mu = \frac{1}{1+\tilde{\varepsilon}}\mu \geq \frac{1}{1+\tilde{\varepsilon}}(1+\tilde{\varepsilon})k = k$ und wir erhalten

$$\begin{aligned} P\left(\deg(v) \leq k\right) &\leq P\left(\deg(v) \leq (1-\delta)\mu\right) \leq \exp\left(-\frac{\delta^2}{2}\mu\right) = \exp\left(-\Theta(k)\right) \\ &= o\left(\frac{1}{k}\right). \end{aligned}$$

Diese Aussage nutzend, können wir die Rekursionsgleichung für T_p vereinfachen:

$$\begin{aligned} T_p(n,k) = T_p(n-1,k-1) &+ \sum_{d_v=1}^{n-1} P\left(deg(v) = d_v\right) \cdot T_p\left(n-1-d_v, k-d_v\right) + \ldots \\ &+ \mathcal{O}(n), \end{aligned}$$

wie wir sie aus dem Algorithmus erhalten: Der erste Summand betrachtet den Fall, dass der Knoten v Teil der zu findenden Lösungsmenge ist, der zweite den Fall, in welchem die d_v Nachbarn von v dies sind.
Zuerst bemerke man, dass alle Summanden, die zu negativen Parametern führen, ignoriert werden können, da sie von dem $\mathcal{O}(n)$-Summand dominiert werden. Dies rechtfertigt, die Summe an der Stelle $d_v = k$ enden zu lassen:

$$\begin{aligned} T_p(n,k) = T_p(n-1,k-1) &+ \sum_{d_v=1}^{k} P\left(deg(v) = d_v\right) \cdot T_p\left(n-1-d_v, k-d_v\right) + \ldots \\ &+ \mathcal{O}(n). \end{aligned} \tag{4.1}$$

Unter der Annahme, dass die gesuchte erwartete Laufzeit monoton in beiden Komponenten ist, erhalten wir weiterhin

$$T_p\left(n-1-d_v, k-d_v\right) \leq T_p\left(n-1-1, k-1\right) \leq T_p\left(n, k-1\right) \text{ für alle } d_v \geq 1.$$

Damit ergibt sich

$$\begin{aligned} T_p(n,k) &\leq T_p(n,k-1) + T_p(n,k-1) \cdot \sum_{d_v=1}^{k} P\left(deg(v) = d_v\right) + \mathcal{O}(n) \\ &\leq T_p(n,k-1) + T_p(n,k-1) \cdot P\left(deg(v) \leq k\right) + \mathcal{O}(n) \\ &\leq T_p(n,k-1) \cdot \left(1 + o\left(\frac{1}{k}\right)\right) + \mathcal{O}(n). \end{aligned} \tag{4.2}$$

Wendet man Gleichung (4.2) rekursiv an, was wegen $n' := n$ und $k' := k-1$ und damit $\frac{(1+\varepsilon)k'}{n'} < \frac{(1+\varepsilon)k}{n} \leq p$ möglich ist, erhält man

$$T_p(n,k) \leq \left(1 + o\left(\frac{1}{k}\right)\right)^k \cdot T_p(n,0) + \mathcal{O}(kn) = \mathcal{O}(kn). \tag{4.3}$$

Tatsächlich sieht man, dass die Funktion $T_p(n,k) := c \cdot kn$ mit einer festen Konstante $c > 0$ sowohl die Rekursion wie auch die Monotonie-Annahme erfüllt. □

Dieser Satz ist nützlich, wenn nur eine kleine Knotenüberdeckung der Größe $k = o(n)$ gesucht wird. Vergrößert man den Parameter k, wird die Wahrscheinlichkeitsverteilung, gemäß derer der Eingabegraph gewählt wird, immer weiter eingeschränkt: Die Kantenwahrscheinlichkeit darf nicht zu schnell gegen 0 gehen. Darf die Knotenüberdeckung, die wir suchen, eine Größe von $k = \Theta(n)$ besitzen, wird diese Forderung nur mit Kantenwahrscheinlichkeiten erfüllt werden, die strikt von 0 weggbeschränkt sind, d.h., deren lim inf echt größer als 0 ist. Im Kontext parametrisierter Komplexität ist man jedoch häufig an Problemstellungen mit kleinen Parametern interessiert und dafür erhalten wir die folgende, starke Aussage:

Korollar 4.4.3. *Sei $k' \colon \mathbb{N} \to \mathbb{N}$, $k < k'(n)$ und $G \sim RG(n,p)$ mit $1 \geq p = \omega\left(\frac{k'(n)}{n}\right)$. Dann ist die erwartete Laufzeit von Algorithmus 4.3 gegeben durch den Ausdruck $T_p(n,k) = \mathcal{O}\left(k \cdot n\right)$.*

Beweis. Ist $k < k'$, so folgt $p = \omega\left(\frac{k'(n)}{n}\right) = \omega\left(\frac{k}{n}\right)$, und Satz 4.4.2 kann angewendet werden, was zum gewünschten Ergebnis führt. □

In einer parametrisierten Auffassung des Problems erhalten wir also eine erwartete Laufzeit unseres Algorithmus, welche linear in k (und n) ist, solange der erwartete Knotengrad schneller wächst als die Größe der gesuchten Knotenüberdeckung.

Dies ist ein starkes Resultat, wenn man es mit den besten bekannten Worst-Case-Ergebnissen von $\mathcal{O}\left(kn + c^k\right)$ mit Konstanten $c > 1$ vergleicht.

Im Beweis von Satz 4.4.2 haben wir ausgenutzt, dass unter den Bedingungen des Satzes jeder Rekursionsaufruf im Schritt 6 von Algorithmus 4.3 direkt zu einem trivialen Teilproblem führt. Diese Idee wird im Folgenden erweitert, dass sich nun nach einer endlichen Zahl solcher Rekursionsaufrufe ein triviales Teilproblem mit negativem Parameter ergibt.

Satz 4.4.4. *Seien $1 \leq \ell \in \mathbb{N}$ und $0 < \varepsilon$ Konstanten sowie $G \sim RG(n,p)$, wobei jetzt $(1+\varepsilon) \cdot k \leq n \cdot \left(1 - (1-p)^\ell\right)$ gelten soll. Dann ist die erwartete Laufzeit von Algorithmus 4.3 gleich $T_p(n,k) = \mathcal{O}\left(k^\ell \cdot n\right)$.*

Beweis. Der Fall $\ell = 1$ ist bereits durch Satz 4.4.2 gegeben. Die übrigen werden hier per Vollständiger Induktion gezeigt. Dafür sei die Korrektheit der Aussage für ℓ vorausgesetzt und wir zeigen sie nun für $\ell + 1$:

In analoger Weise wie im Beweis zu Satz 4.4.2 erhalten wir für den gewählten Knoten v, dass sein erwarteter Knotengrad $\mu \geq pn$ ist. Deshalb folgt mit $0 < \delta := 1 - \frac{1}{1+\tilde{\varepsilon}} < 1$ wie oben die Abschätzung

$$\begin{aligned} P\left(\deg(v) \leq \frac{1}{1+\varepsilon} pn\right) &\leq P\left(\deg(v) \leq (1-\delta)\mu\right) \leq \exp\left(-\frac{\delta^2}{2}\mu\right) \\ &= \exp\left(-\Theta(k)\right) = o\left(\frac{1}{k}\right). \end{aligned}$$

Dies nutzend, vereinfacht sich die Rekursionsgleichung (4.1) zu

$$\begin{aligned} T_p(n,k) = T_p(n-1,k-1) &+ \sum_{d_v=1}^{\frac{1}{1+\varepsilon}pn} P\left(deg(v) = d_v\right) \cdot T_p\left(n-1-d_v, k-d_v\right) \\ &+ \sum_{d_v=\frac{1}{1+\varepsilon}pn+1}^{n} P\left(deg(v) = d_v\right) \cdot T_p\left(n-1-d_v, k-d_v\right) \\ &+ \mathcal{O}(n). \end{aligned}$$

Wieder unter Annahme der Monotonie in beiden Komponenten lässt sich dies weiter vereinfachen:

$$\begin{aligned} T_p(n,k) \leq T_p(n,k-1) + T_p\left(n,k-1\right) &\cdot \sum_{d_v=1}^{\frac{1}{1+\varepsilon}pn} P\left(deg(v) = d_v\right) \\ &+ T_p\left(n',k'\right) \cdot \sum_{d_v=\frac{1}{1+\varepsilon}pn+1}^{n} P\left(deg(v) = d_v\right) + \mathcal{O}(n) \end{aligned}$$

und weiter zu

$$\begin{aligned} T_p(n,k) &\leq T_p(n,k-1) + T_p\left(n,k-1\right) \cdot P\left(\deg(v) \leq \frac{1}{1+\varepsilon}pn\right) + T_p\left(n',k'\right) + \mathcal{O}(n) \\ &\leq \left(1 + o\left(\frac{1}{k}\right)\right) T_p(n,k-1) + T_p\left(n',k'\right) + \mathcal{O}(n)\ , \end{aligned}$$

wobei $n' := n - \frac{1}{1+\varepsilon}pn < n$ und $k' := k - \frac{1}{1+\varepsilon}pn$ ist.

Beim Löschen der Knoten fällt deren Anzahl immer mindestens um den gleichen Wert wie der Parameter, aber im hier betrachteten Bereich immer um mindestens $\frac{1}{1+\varepsilon}pn$. Damit ist

$$\begin{aligned} (1+\varepsilon)k' &\leq (1+\varepsilon)\left(k - \frac{1}{1+\varepsilon}pn\right) = (1+\varepsilon)k - pn \leq n \cdot (1-(1-p)^{\ell+1}) - pn \\ &= (n-pn)\cdot(1-(1-p)^{\ell}) < n' \cdot (1-(1-p)^{\ell}). \end{aligned}$$

Also kann die Induktionsvoraussetzung für $T_p\left(n',k'\right)$ angewandt werden:

$$\begin{aligned} T_p(n,k) &\leq \left(1 + o\left(\frac{1}{k}\right)\right) T_p(n,k-1) + \mathcal{O}\left(k^{\ell} \cdot n\right) + \mathcal{O}(n) \\ &= \left(1 + o\left(\frac{1}{k}\right)\right) T_p(n,k-1) + \mathcal{O}\left(k^{\ell} \cdot n\right). \end{aligned}$$

Dem Beweis von Satz 4.4.2 weiter folgend, kann das Ergebnis rekursiv in den ersten Summanden eingesetzt werden, denn es ist $\frac{k-1}{n-1} < \frac{k}{n}$ und damit alle Voraussetzungen des Satzes für diesen Rekursionsaufruf weiterhin erfüllt. Wir erhalten dadurch

$$T_p(n,k) \leq \left(1 + o\left(\frac{1}{k}\right)\right)^k \cdot T_p(n,0) + \mathcal{O}(k^{\ell+1}n) = \mathcal{O}(k^{\ell+1}n), \tag{4.4}$$

was den Fall $\ell+1$ und dadurch induktiv für alle $\ell \in \mathbb{N}$ zeigt. Auch die angenommene Monotonie wird dabei erfüllt. □

Korollar 4.4.5. *Seien $0 < p < 1$ und $0 < \tilde{\varepsilon} < 1$ Konstanten, $G \sim RG(n,p)$ und $k < (1-\tilde{\varepsilon})n$. Dann ist es möglich, in polynomieller erwarteter Laufzeit eine Knotenüberdeckung der Größe höchstens k zu finden bzw. deren Nichtexistenz zu beweisen.*

Beweis. Da p eine Konstante ist, gibt es eine natürliche Zahl ℓ mit $\frac{k}{n} < (1-\tilde{\varepsilon}) < (1-p)^{\ell}$. Setzt man dieses ℓ in Satz 4.4.4 ein, kann dieser angewendet werden und man erhält eine erwartete Laufzeit von $\mathcal{O}\left(k^{\ell}n\right) = \mathcal{O}\left(n^{\ell+1}\right)$ für den Algorithmus 4.3, der eine Knotenüberdeckung der Größe höchstens k findet, oder deren Nichtexistenz nachweist. Iteriert man den Parameter k von 1 bis $(1-\tilde{\varepsilon})n$, bis man eine Lösung gefunden oder das Limit erreicht hat, erhält man damit einen Algorithmus mit erwarteter Laufzeit $\mathcal{O}\left(n^{\ell+2}\right)$, der das gewünschte Ergebnis liefert. □

In Satz 4.4.2 betrachteten wir den Fall, dass mit hoher Wahrscheinlichkeit der Rekursionsaufruf bei Ausschluss des betrachteten Knotens v direkt zu einem trivialen „Restproblem“ mit negativem Parameter führt. In Satz 4.4.4 gingen wir in ähnlicher Weise davon aus, dass dies nach endlich vielen solchen Aufrufen mit hoher Wahrscheinlichkeit der Fall ist. Nun soll untersucht werden, wie sich der Algorithmus verhält, wenn diese Anzahl unbeschränkt ist:

Satz 4.4.6. *Sei $f\colon \mathbb{N} \to \mathbb{R}^+$ mit $f(n) = \omega\left(\frac{1}{n}\right)$ eine monoton fallende Funktion und $G \sim RG(n,p)$ mit $(1+\varepsilon)\, f(n) < p(n)$ für alle genügend großen n. Dann ist die erwartete Laufzeit von Algorithmus 4.3 gleich $T_p(n,k) = n^{\mathcal{O}\left(\frac{\log n}{f(n)}\right)}$.*

Beweis. Wie in den vorangegangenen Beweisen schließen wir, dass nur mit einer Wahrscheinlichkeit von $o\left(\frac{1}{n}\right)$ der Grad des betrachteten Knoten kleiner ist als der Ausdruck $f(n) \cdot n$. Damit vereinfacht sich die Rekursionsgleichung unter Annahme der Monotonie in beiden Komponenten zu

$$T_p(n,k) \leq \left(1 + o\left(\frac{1}{n}\right)\right) T_p(n-1,k-1) + T_p\left(n \cdot (1-f(n)), k \cdot (1-f(n))\right) + \ldots + \mathcal{O}(n).$$

Wieder lässt sich die Rekursion in ihren ersten Summanden einsetzen. Iteriert man dies und nutzt die Monotonie aus, geht diese über in

$$T_p(n,k) \leq n \cdot (1-f(n)) \cdot T_p\left(n \cdot (1-f(n)), k \cdot (1-f(n))\right) + \mathcal{O}\left(n^2\right).$$

Der besseren Lesbarkeit wegen substituieren wir $1 - f(n)$ durch $g(n)$. Dann ist auch $0 < g(n) < 1$ und die Rekursionsgleichung vereinfacht sich zu

$$T_p(n,k) \leq n \cdot g(n) \cdot T_p\left(n \cdot g(n), k \cdot g(n)\right) + \mathcal{O}\left(n^2\right). \tag{4.5}$$

Wir wollen diesen Prozess des Einsetzens so lang iterieren, bis der T_p-Teil trivial wird. In jeder Iteration erhalten wir einen zusätzlichen Faktor n, weshalb wir eine möglichst kleine Anzahl solcher Einsetzungen verwenden wollen.

Da f monoton fallend ist, ist g monoton steigend und nimmt damit für kleinere Argumente auch kleinere Funktionswerte an. Also ist $g(n \cdot g(n)) \leq g(n)$. Das ℓ-mal wiederholte Einsetzen von Gleichung (4.5) in sich selbst kann dann aufgrund dieser Monotonie-Aussagen vereinfacht werden zu

$$T_p(n,k) \leq n^\ell \cdot g(n)^\ell \cdot T_p\left(n \cdot g(n)^\ell, k \cdot g(n)^\ell\right) + \mathcal{O}\left(\ell \cdot n^2\right).$$

Wählen wir ℓ als $\frac{1}{f(n)}$, dann ist $g(n)^\ell = \left(1 - \frac{1}{1/f(n)}\right)^{1/f(n)}$, was gegen $e^{-1} < \frac{1}{2}$ konvergiert. Also ist

$$T_p(n,k) \leq n^{1/f(n)} \cdot T_p\left(n',k'\right) + \mathcal{O}\left(n^3\right) \text{ mit } \frac{n'}{n} = \frac{k'}{k} = \frac{1}{2}\,.$$

Nach höchstens $\frac{1}{f(n)}$ Iterationen der Rekursionsgleichung (4.5) wird dabei demnach die Anzahl n der Knoten halbiert. Für die nächste Halbierung braucht es dann $\frac{1}{f(n/2)} \leq \frac{1}{f(n)}$ Iterationen und so weiter. Um die Knotenanzahl so weit zu verringern, dass der T_p-Anteil trivial wird, benötigt es $\mathcal{O}(\log n)$ solcher Halbierungen. Es folgt, dass dies also nach $\mathcal{O}\left(\frac{\log n}{f(n)}\right)$ Iterationen der Rekursionsgleichung (4.5) der Fall ist. Daraus erhalten wir das gewünschte Ergebnis

$$T_p(n,k) \leq n^{\mathcal{O}\left(\frac{\log n}{f(n)}\right)}.$$

□

Korollar 4.4.7. *In der Situation von Satz 4.4.6 sei $f(n) = \Omega\left(n^{-c}\right)$ mit einer Konstanten $0 < c < 1$. Dann ist $T_p(n,k) = n^{\mathcal{O}(n^c)}$. Für $f(n) = n^c$ beträgt die erwartete Laufzeit also $T_p(n,k) = n^{\mathcal{O}\left(\frac{1}{f(n)}\right)}$.*

Beweis. Zuerst bemerken wir, dass die erwartete Laufzeit des Algorithmus monoton fallend in f ist. Deshalb genügt es, wenn wir uns auf den Spezialfall $f(n) = n^{-c}$ beschränken.

Wie im Beweis von Satz 4.4.6 erhalten wir eine Halbierung der Knotenanzahl n nach höchstens $\frac{1}{f(n)}$ Iterationen, in der wir jedesmal die Rekursionsgleichung (4.5) in ihren ersten Summanden einsetzen. Im hier betrachteten Fall haben wir nun nicht nur $\frac{1}{f(n/2)} \leq \frac{1}{f(n)}$ wie in der allgemeineren Situation des Satzes 4.4.6, sondern genauer $\frac{1}{f(n/2)} = \left(\frac{n}{2}\right)^c \leq d \cdot \frac{1}{f(n)}$ mit einer Konstanten $d := 2^{-c}$. Offensichtlich ist $0 < d < 1$. Damit lässt sich die Gesamtzahl der benötigten Iterationen nun besser abschätzen, da die Summe

$$\frac{1}{f(n)} + \frac{1}{f(n/2)} + \frac{1}{f(n/4)} + \cdots \leq \frac{1}{f(n)}\left(1 + d + d^2 + \ldots\right) = \frac{1}{f(n)} \cdot \frac{1}{d-1}$$

gegen ein konstantes Vielfaches von $\frac{1}{f(n)}$ „konvergiert“. Also ist $T_p(n,k) = n^{\mathcal{O}\left(\frac{1}{f(n)}\right)}$.

□

Satz 4.4.6 und Korollar 4.4.7 zeigen, dass für verschiedene Parameter-Bereiche, z. B. konstante Kantenwahrscheinlichkeiten $p > 0$ oder auch deutlich kleinere bis zu $\omega\left(\frac{\log^2 n}{n}\right)$, die erwartete Laufzeit des beschränkten Suchbaum-Algorithmus 4.3 subexponentiell ist.

Weiterhin können wir diese Resultate auch im Kontext einer Smoothed-Analyse wiedergeben: Um dies zu tun, interpretieren wir die verschiedenen Kantenwahrscheinlichkeiten in unserem Zufallsgraph-Model als das Ergebnis des „Verrauschens“ eines gegebenen Graphen. War in diesem Graphen eine bestimmte Kante enthalten, ist es diese im Zufallsgraphen nur noch mit Wahrscheinlichkeit $1 - \varepsilon$. Und war die Kante nicht da, wird sie nun mit Wahrscheinlichkeit ε hinzugefügt.

In einer analogen Weise wie in Korollar 4.4.5 erhalten wir für konstante Stör-Wahrscheinlichkeiten ε den folgenden Satz.

Satz 4.4.8 (Smoothed-Komplexität für das Vertex-Cover-Problem mit konstanter Stör-Wahrscheinlichkeit)**.** *Seien $n \in \mathbb{N}$ und $G = (V, E)$ ein Graph mit $n := |V|$ Knoten. Weiterhin seien $0 < \varepsilon \leq \frac{1}{2}$ und $0 < \tilde{\varepsilon}$ Konstanten sowie $k < (1-\tilde{\varepsilon})n$. Dann besitzt der beschränkte Suchbaum-Algorithmus 4.3 eine polynomielle parametrisierte Smoothed-Komplexität von $SC(n, k, \varepsilon)$.*

Beweis. Sei G ein beliebiger Graph mit n Knoten. Dann ist $RG(G, \varepsilon)$ nach Konstruktion eine ε-beschränkte Wahrscheinlichkeitsverteilung auf der Menge der Graphen mit n Knoten. Damit kann Korollar 4.4.5 angewendet werden und man erhält eine polynomielle erwartete Laufzeit. Nimmt man dann das Supremum über alle $G \in \mathcal{G}_n$, so ändert dies nichts, da diese Abschätzung an die erwartete Laufzeit nicht mehr von G abhängt, was den Beweis erfolgreich abschließt. □

Damit erhalten wir eine polynomielle Laufzeit für die Suche nach einer Minimum-Knotenüberdeckung der Größe maximal $(1 - \tilde{\varepsilon})n$, wenn wir ein konstantes Verrauschen, d. h. Stör-Wahrscheinlichkeit, zulassen. Darüber hinaus kann man sich dabei auch auf solches Rauschen beschränken, welches keine Kante aus dem Graphen G löscht (diese also alle sicher erhalten bleiben).

Im nächsten Schritt lassen wir wie in Satz 4.4.6 und Korollar 4.4.7 dieses Rauschen gegen 0 gehen:

Satz 4.4.9 (Smoothed-Komplexität für Vertex-Cover mit gegen 0 gehender Stör-Wahrscheinlichkeit)**.** *Sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten. Weiterhin sei $\frac{1}{2} \geq \varepsilon(n) = \omega\left(\frac{n^c}{n}\right)$ eine Stör-Wahrscheinlichkeit, wobei $0 < c < 1$ eine Konstante ist. Dann hat der iterierte beschränkte Suchbaum-Algorithmus, den man aus Algorithmus 4.3 durch Iterieren des Parameters k von 1 bis n erhält, eine subexponentielle Smoothed-Komplexität von $SC(n, \varepsilon) = \exp\left(\mathcal{O}\left(\frac{\log n}{\varepsilon}\right)\right)$.*

Beweis. Für jedes solche k hat Algorithmus 4.3 eine erwartete Laufzeit von

$$T_p(n, k) = n^{\mathcal{O}\left(\frac{1}{\varepsilon(n)}\right)} = \exp\left(\mathcal{O}\left(\frac{\log n}{\varepsilon}\right)\right),$$

wie es in Korollar 4.4.7 gezeigt wurde. Iteriert man dies über alle $0 < k \leq n$, erhält man einen zusätzlichen Faktor n in der Laufzeit. Aber dies ändert den Term nicht, da wegen $\varepsilon \leq \frac{1}{2}$ und damit $\frac{1}{\varepsilon} \geq 2$ schließlich $\log n = \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$ ist. □

Korollar 4.4.10. *Unter den gleichen Voraussetzungen wie in Satz 4.4.9 gelte für die Fehlerfunktion* $\frac{1}{2} \geq \varepsilon(n) = \omega\left(\frac{\log^2 n}{n}\right)$. *Dann ist die Smoothed-Komplexität* $SC(n, \varepsilon)$ *des iterierten beschränkten Suchbaum-Algorithmus subexponentiell und lässt sich angeben durch* $\exp\left(o\left(\frac{\log^2 n}{\varepsilon}\right)\right)$.

Beweis. In dieser etwas allgemeineren Situation wie in Satz 4.4.9 können wir nun den Satz 4.4.6 anwenden. Zuerst bemerken wir aber, dass es eine Funktion $f(n)$ mit $f(n) = \omega\left(\frac{\log^2 n}{n}\right)$ und $\varepsilon(n) = \omega(f(n))$ gibt. Da die erwartete Laufzeit monoton fallend in der Fehler- bzw. Wahrscheinlichkeitsfunktion ist, können wir statt $\varepsilon(n)$ zur Gewinnung einer oberen Schranke auch $f(n)$ verwenden.

Auf analoge Weise wie im Beweis von Satz 4.4.9 erhalten wir damit dann eine Smoothed-Komplexität von $SC(n, \varepsilon) = \exp\left(\mathcal{O}\left(\frac{\log^2 n}{f(n)}\right)\right) = \exp\left(o\left(\frac{\log^2 n}{\varepsilon}\right)\right)$. □

Zusammengefasst haben wir also das Ergebnis: Selbst, wenn nur eine genügend große Nullmenge der Kanten (jedoch mehr als ein Anteil von $\frac{\log^2 n}{n}$) gestört wird, sinkt die erwartete Laufzeit bei der Suche nach einer Minimum-Knotenüberdeckung auf eine subexponentielle Größe in Abhängigkeit der Knotenanzahl n des betrachteten Graphen.

4.5 Das CLIQUE-Problem

Das (parametrisierte) CLIQUE-Problem kann man als folgende Fragestellung definieren: Gegeben sei ein Graph $G = (V, E)$ und eine natürliche Zahl k. Gibt es eine Teilmenge $S \subset V$ von Knoten aus G der Größe k, welche einen vollständigen Teilgraphen induziert?

Um dieses Problem zu lösen, bedienen wir uns wieder der Methode des beschränkten Suchbaums. Dazu beobachten wir, dass wir einen Knoten v, der nicht zur Lösungsmenge gehört, einfach (mit all seinen Kanten) löschen können. Und ist er in S, können wir den Graphen G auf den Teilgraphen, der durch die Nachbarn von v induziert wird, einschränken (sowie gleichzeitig den Parameter k um 1 herunterzählen, da ja v als Teil der Lösungsmenge ausgewählt wurde). Damit erhält unser Suchbaum-Algorithmus die Gestalt, wie sie in Algorithmus 4.4 angegeben ist.

Sei wie oben $T_{p,q}$ die erwartete Laufzeit dieses Algorithmus für (p, q)-beschränkte Verteilungen der Eingabe-Graphen und d_v der Grad des gerade betrachteten Knotens v. Dann erhalten wir für die Laufzeit gemäß dem Algorithmus die Rekursion

$$T_{p,q}(n, k) \leq T_{p,q}(n-1, k) + T_{p,q}(d_v, k-1) + \mathcal{O}(n); \text{ für } n \geq k > 0 \text{ und} \quad (4.6)$$
$$T_{p,q}(n, k) = \mathcal{O}(1); \text{ für } k \leq 0 \text{ oder } n < k.$$

Algorithmus 4.4 : Einfacher beschränkter Suchbaum-Algorithmus für das CLIQUE-Problem

```
// Algorithmus 4.2, in dem die Schritte 2 bis 6′ durch die
   folgenden ersetzt wurden:
```
2 Nimm zuerst an, dass v Teil der Lösungsmenge ist: Schränke G auf die Nachbarschaft von v ein und löse das Problem rekursiv mit Parameter $k' = k - 1$.
3 **if** *Wurde eine Lösung gefunden?* **then**
4 | **return** *„ja“ und die Lösungsmenge vereinigt mit* $\{v\}$ *selbst.*
5 **else** `// Wenn keine Lösung mit` v `gefunden wurde, suche nach einer ohne diesen Knoten.`
6 | Lösche v und löse das Problem rekursiv mit gleichem Parameter k (aber einem Knoten weniger).
7 | **if** *Wurde eine Lösung gefunden?* **then return** *„ja“ und die Lösungsmenge selbst.*
8 **return** *„nein“.*

Geben wir die Rekursion für $T_{p,q}$ etwas genauer an, erhalten wir für $n \geq k > 0$ die Beziehung

$$T_{p,q}(n,k) \leq T_{p,q}(n-1,k) + \sum_{d_v=0}^{n-1} P\left(\deg(v) = d_v\right) \cdot T_{p,q}\left(d_v, k-1\right) + \mathcal{O}(n). \tag{4.7}$$

Sei $\alpha_{p,q}(n,k)$ die Wahrscheinlichkeit, dass es in einem (p,q)-beschränkten Zufallsgraphen mit n Knoten keine k-Clique gibt. Dann gibt der Aufruf in Schritt 2 von Algorithmus 4.4 eine „nein“-Antwort nur mit Wahrscheinlichkeit $\alpha_{p,q}\left(d_v, k-1\right)$ zurück. Und nur in diesem Fall, dass noch keine Lösung gefunden wurde, muss die Berechnung in Schritt 6 durchgeführt werden.

Offensichtlich ist $\alpha_{p,q} \leq 1$, was zur Rekursionsgleichung (4.7) führt. Verwenden wir aber eine genauere Abschätzung für $\alpha_{p,q}$, gibt die folgende Version der Rekursion bessere und schärfere Resultate zur erwarteten Laufzeit des Algorithmus:

$$\begin{aligned} T_{p,q}(n,k) \leq & \sum_{d_v=0}^{n-1} P(\deg(v)=d_v) \cdot \left(T_{p,q}\left(d_v, k-1\right) + \alpha_{p,q}\left(d_v, k-1\right) \cdot T_{p,q}(n-1,k)\right) \\ & + \mathcal{O}(n); \text{ für } n \geq k > 0\ . \end{aligned} \tag{4.8}$$

Nun möchten wir diese Rekursionsgleichungen ähnlich wie im Abschnitt 4.4 analysieren. Dabei betrachten wir zunächst die durch Gleichung (4.7) gegebene Rekursion und erhalten eine erste Abschätzung an die erwartete Laufzeit von Algorithmus 4.4.

Satz 4.5.1. *Seien* $\delta\colon \mathbb{N} \to \mathbb{R}^+$ *mit* $\delta = \omega\left(\sqrt{\frac{\ln n}{n}}\right)$ *und* $q < 1 - \delta$*. Weiterhin sei* $G \sim RG(n, p, q')$ *mit* $0 \leq p \leq q' \leq (1-\delta)q < 1$ *. Dann ist die erwartete Laufzeit von Algorithmus 4.4 gleich* $T_{p,q}(n,k) = \mathcal{O}\left(\exp\left(\frac{\log^2 n}{-\log q}\right)\right) = n^{\mathcal{O}\left(\frac{\log n}{-\log q}\right)}$*. Ist* q *wegbeschränkt von 1 (d. h.* $\limsup q(n) < 1$*), dann liegt die erwartete Laufzeit in* $\exp\left(\mathcal{O}\left(\log^2 n\right)\right) = n^{\mathcal{O}(\log n)}$*.*

Beweis. Zuerst berechnen wir die Wahrscheinlichkeit, dass der betrachtete Knoten v einen Grad $d_v > qn$ besitzt. Wie im Beweis von Satz 4.4.2 nutzen wir die Chernoff-Schranke, wie sie in Lemma 2.8.1 gegeben ist, und erhalten, dass diese gesuchte Wahrscheinlichkeit in $\exp\left(-\frac{1}{2}\delta^2 qn\right) = o\left(\frac{1}{n}\right)$ liegt. Damit und unter Annahme von Monotonie geht die Gleichung (4.7) über in

$$T_{p,q}(n,k) \leq \left(1 + o\left(\frac{1}{n}\right)\right) \cdot T_{p,q}(n-1,k) + T_{p,q}\left(qn, k-1\right) + \mathcal{O}(n).$$

Iteriert man diese Rekursion und nutzt weiter die Monotonie aus, erhält man

$$\begin{aligned} T_{p,q}(n,k) &\leq \left(1 + o\left(\frac{1}{n}\right)\right)^n \cdot T_{p,q}(0,k) + n \cdot T_{p,q}\left(qn, k-1\right) + \mathcal{O}\left(n^2\right) \\ &\leq n \cdot T_{p,q}\left(qn, k-1\right) + \mathcal{O}\left(n^2\right). \end{aligned}$$

Iteriert man nun diese erhaltene Rekursion $-\frac{\log n}{\log q}$ mal, sodass die q-Potenz kleiner wird als $\frac{1}{n}$, gibt dies die gewünschte Abschätzung:

$$\begin{aligned} T_{p,q}(n,k) &\leq n^{-\frac{\log n}{\log q}} \cdot T_{p,q}(1,k') + \mathcal{O}\left(n^2 \log n\right) \\ &= \mathcal{O}\left(\exp\left(\frac{\log^2 n}{-\log q}\right)\right) = n^{\mathcal{O}\left(\frac{\log n}{-\log q}\right)}. \end{aligned}$$

Und ist q kleiner als eine Konstante < 1, liegt dies in

$$= \exp\left(\mathcal{O}\left(\log^2 n\right)\right) = n^{\mathcal{O}(\log n)}. \qquad \square$$

Im nächsten Satz soll die schärfere Rekursionsgleichung (4.8) betrachtet werden. Um bessere Abschätzungen für $\alpha_{p,q}(n,k)$ zu erhalten, beschränken wir uns dabei zuerst auf konstante Parameter k:

Lemma 4.5.2. *Seien* $k \in \mathbb{N}$ *eine Konstante sowie* $1 > q \geq p\colon \mathbb{N} \to \mathbb{R}$ *reellwertige Funktionen. Sei weiterhin* $G \sim RG(n,p,q)$*. Ist* $\log p = -o(\log n)$*, dann gilt für (abhängig von* k*) alle genügend großen* n*, dass* $\alpha_{p,q}(n,k) < \exp\left(-n^{\frac{1}{2}}\right)$ *ist.*

Beweis. Sei P die Wahrscheinlichkeit, dass k beliebige, aber konkret ausgewählte Knoten in G eine k-Clique bilden. Dann ist $P \geq p^{\binom{k}{2}} = \exp\left(\log p \cdot \frac{k(k-1)}{2}\right)$. Es gibt $\frac{n}{k} = \Theta(n)$ paarweise disjunkte Knotenteilmengen der Größe k im Graphen G. Damit hat jede solche Teilmenge, unabhängig von den übrigen, die Wahrscheinlichkeit von höchstens $1-P < \exp(-P)$ keine k-Clique zu bilden. Also ist die Wahrscheinlichkeit $\alpha_{p,q}(n,k)$, dass G keine k-Clique enthält, höchstens $\exp(-P \cdot \Theta(n))$. Nun bemerken wir, dass mit $\log p$ auch $\log P$ in $-o(\log n)$ liegt und damit $P > n^{-\frac{1}{2}}$ für alle genügend großen n gilt. Damit ist schließlich $\alpha_{p,q}(n,k) < \exp\left(-n^{\frac{1}{2}}\right)$ für alle $n > n_0$, wobei n_0 von k abhängt. □

Dieses Lemma ist weitgehend identisch mit Lemma 6 aus [43]. Es stellt ein effektives Werkzeug zur Vereinfachung der Rekursionsgleichung (4.8) dar, falls die untere Schranke an die Kantenwahrscheinlichkeit nicht zu klein ist. (So kann z. B. p als positiv-konstant oder $p = \omega\left(n^{-\beta}\right)$ für beliebiges, konstantes $\beta > 0$ gewählt werden.) In dieser Situation erhält man $\log \alpha_{p,q}(n,k) < -n^{\frac{1}{2}}$, was beim Beweis des folgenden Satzes benutzt werden wird.

Satz 4.5.3. *Sei $\delta\colon \mathbb{N} \to \mathbb{R}^+$ mit $\delta = \omega\left(\sqrt{\frac{\ln n}{n}}\right)$. Weiterhin seien $\log p = -o(\log n)$ und $q < 1 - \delta$. Darüber hinaus sei $G \sim RG(n, p', q')$ mit $(1+\varepsilon)\, p \leq p' \leq q' \leq (1-\delta)q$ für eine Konstante $0 < \varepsilon < 1$. Dann gibt es eine Funktion $k'\colon \mathbb{N} \to \mathbb{N}$ mit $k' = \omega(1)$, sodass für alle $k < k'$ der Algorithmus 4.4 eine erwartete Laufzeit von $T_{p,q}(n,k) = \mathcal{O}(k \cdot n)$ besitzt. Dabei hängt die $\mathcal{O}$-Konstante nicht von k ab.*

Beweis. In diesem Beweis werden wir die Aussage über die Laufzeit für alle festgewählten k und alle genügend großen n (wobei die untere Schranke von k abhängt) zeigen. Dabei sei darauf hingewiesen, dass die $\mathcal{O}$-Konstanten (also konstante Faktoren, die durch dieses Symbol unterdrückt werden), die in dieser Berechnung implizit Verwendung finden, nicht vom Parameter k abhängen.

Damit ist dies äquivalent zur Aussage, dass es eine monoton und unbeschränkt wachsende Funktion k' existiert, sodass die im Satz gemachte Aussage für alle n und $k < k'(n)$ gezeigt wird. Die Funktion k' geben wir hierbei jedoch nicht explizit an.

Aufgrund dieser Beobachtung können wir im weiteren Verlauf dieses Beweises den Parameter k als „Konstante“ betrachten und führen eine Induktion über diesen aus. Für $k = 0$ gibt es nichts zu zeigen. Deshalb sei nun $k \geq 1$ und wir nehmen die Aussage für $k-1$ als schon gezeigt an.

Um die Summe in der Rekursionsgleichung (4.8) berechnen zu können, nutzen wir Lemma 4.5.2 und ersetzen also den Koeffizienten $\alpha_{p,q}(d_v, k-1)$ durch die im Lemma erhaltene Abschätzung von $\exp\left(-n^{\frac{1}{2}}\right)$.

Damit ist

$$T_{p,q}(n,k) < \sum_{d_v=pn+1}^{n-1} P\left(\deg(v)=d_v\right) \cdot \Big(T_{p,q}\left(d_v, k-1\right) + \exp\left(-n^{\frac{1}{2}}\right) \cdot T_{p,q}(n-1,k) \Big) + \mathcal{O}(n).$$

Nun benutzen wir $T_{p,q}(n-1,k) = \exp \mathcal{O}\big(\log^2 n\big)$ aus Satz 4.5.1. Offensichtlich reduziert sich das Produkt mit dem α-Term zu einem vernachlässigbarem $o(1)$, was durch die übrigen Terme dominiert wird. Die Summe reduziert sich also auf

$$< \sum_{d_v=q^{\frac{1}{2-\varepsilon}}n+1}^{n-1} P\left(\deg(v) = d_v\right) \cdot T_{p,q}\left(d_v, k-1\right) + o(n).$$

Zur Abschätzung von $T_{p,q}\left(d_v, k-1\right)$ verwenden wir die Induktions-Voraussetzung und erhalten $T_{p,q}\left(d_v, k-1\right) = \mathcal{O}\left((k-1)d_v\right)$, also

$$T_{p,q}(n,k) = \mathcal{O}((k-1)n) + \mathcal{O}(n) = \mathcal{O}(kn)\ ,$$

wobei die $\mathcal{O}$-Konstante unabhängig von k ist. Damit ist der Beweis erbracht. □

Satz 4.5.4 (Das CLIQUE-Problem liegt in avg−FPT)**.** *Sei $\delta\colon \mathbb{N} \to \mathbb{R}^+$ eine Funktion mit $\delta = \omega\left(\sqrt{\frac{\ln n}{n}}\right)$. Weiterhin seien $\log p = -o(\log n)$ und $q < 1-\delta$. Schließlich sei $G \sim RG(n, p', q')$ mit $(1+\varepsilon)\, p \leq p' \leq q' \leq (1-\delta)\, q$ für eine Konstante $0 < \varepsilon < 1$. Dann hat der Algorithmus 4.4 eine erwartete FPT-Laufzeit: Es existiert eine Funktion $f\colon \mathbb{N} \to \mathbb{N}$, die nur vom Parameter k abhängt, sodass der Algorithmus 4.4 eine erwartete Laufzeit von $T_{p,q}(n,k) = f(k) \cdot n$ besitzt.*

Beweis. In Satz 4.5.3 wurde die Existenz einer Konstanten $c_1 > 0$ gezeigt, sodass für jedes k und genügend großes n der Algorithmus eine erwartete Laufzeit von

$$T_{p,q}(n,k) < c_1 k \cdot n \tag{4.9}$$

besitzt. Liest man dies in umgekehrter Richtung, so können wir sagen, dass es eine Funktion $i(n)\colon \mathbb{N} \to \mathbb{N}$ gibt, welche das maximale k für dieses n angibt, sodass die Ungleichung (4.9) noch wahr ist. Dann wissen wir nach dem oben gezeigten, dass $i(n)$ unbeschränkt wächst. Insbesondere gilt also Satz 4.5.4 damit nicht nur für konstante k, sondern für alle $k \leq i(n)$.

Ist dabei $i\left(n'\right)$ für ein n' undefiniert, sind wir fertig. Denn dann gilt die Ungleichung (4.9) für alle k und $n \geq n'$. Im anderen Fall können wir o. B. d. A. annehmen, dass $i(n)$ monoton wächst. Nun definieren wir $u\colon \mathbb{N} \to \mathbb{N}$ als $u = u(k)\colon \mathbb{N} \to \mathbb{N}$,

$u := \max(n, i(n) = k)$. Nach den zuvor gemachten Bemerkungen ist diese Funktion wohldefiniert, monoton wachsend und es ist $u(i(n)) \geq n$ für alle n.

Als nächstes wissen wir aus Satz 4.5.1, dass es eine Konstante $c_2 > 0$ gibt, sodass

$$T_{p,q}(n,k) < T(n) := \exp\left(c_2 \cdot \frac{\log^2 n}{-\log q}\right) \tag{4.10}$$

gilt. Dabei ist $T = T(n)\colon \mathbb{N} \to \mathbb{R}^+$ eine Funktion, die nicht von k abhängt.

Schließlich definieren wir nun $f = f(k)\colon \mathbb{N} \to \mathbb{R}^+$,

$$f(k) := \max\left(c_1 \cdot k, T(u(k))\right).$$

Für diese Funktion f haben wir einerseits für $k \leq l(n)$ die Ungleichung

$$T_{p,q}(n,k) < c_1 k \cdot n \leq f(k) \cdot n$$

und andererseits für $k > l(n)$ nun auch

$$T_{p,q}(n,k) < T(n) \leq T(u(l(n))) \leq T(u(k)) \leq f(k) \leq f(k) \cdot n.$$

Damit erhalten wir also zusammen für alle k die zu zeigende Aussage

$$T_{p,q}(n,k) < f(k) \cdot n.$$

□

Wir merken an, dass die in diesem Beweis benutzte Technik (Abschätzung der Laufzeit für alle k und alle genügend großen n sowie nachfolgender Konstruktion einer Funktion f, die schnell genug wächst, um auch die größeren k abzudecken) auch in anderen und allgemeineren Umständen anwendbar ist, also nicht speziell nur für dieses Problem greift.

Satz 4.5.4 verallgemeinert dabei das Ergebnis von Fountoulakis et al. [43] für einen Bereich von Kantenwahrscheinlichkeiten, die nicht zu nahe an 0 oder 1 liegen. Die Autoren des zitierten Papers konnten die Zugehörigkeit des Problems zu avg−FPT zeigen, wenn der Eingabegraph gemäß der Verteilung des Erdős-Rényi-Modells gezogen wurde. Mit dem hier gezeigten Resultat können nun große Unterschiede in den individuellen Kantenwahrscheinlichkeiten zugelassen werden, d. h., insbesondere anders als im Erdős-Rényi-Modell dürfen sie sich unterscheiden. Man bemerke, dass dabei sowohl „ja“- wie auch „nein“- Instanzen mit hoher Wahrscheinlichkeit vorkommen.

Dadurch, dass die individuellen Kantenwahrscheinlichkeiten sich unterscheiden dürfen, erhalten wir wieder ein Ergebnis der Smoothed-Komplexität:

Satz 4.5.5 (Smoothed-Komplexität für das parametrisierte Clique-Problem)**.** *Es sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten und $1 > \varepsilon(n)$ mit $\log \varepsilon(n) = -o(\log n)$ die Stör-Wahrscheinlichkeit. Dann hat das parametrisierte* Clique*-Problem eine Smoothed-Komplexität von $SC(n, k, \varepsilon) \leq f(k) \cdot n$. Weiterhin existiert eine Funktion $k' \colon \mathbb{N} \to \mathbb{N}$ mit $k' = \omega(1)$, sodass die Smoothed-Komplexität des parametrisierten* Clique*-Problems $SC(n, k, \varepsilon) = \mathcal{O}(kn)$ für alle $k < k'$ ist.*

Beweis. Wie im Beweis von Satz 4.4.8 betrachten wir den Wahrscheinlichkeitsraum $RG(G, \varepsilon)$ über Graphen mit n Knoten und Kantenwahrscheinlichkeiten $p = \varepsilon$ für alle Kanten in G sowie $q = 1 - \varepsilon$ für alle übrigen. Einen Zufallsgraphen gemäß $RG(G, \varepsilon)$ zu ziehen, heißt also jede potentielle Kante in G unabhängig mit Wahrscheinlichkeit ε zu „flippen“. Dies erfüllt die Voraussetzungen der Sätze 4.5.3 und 4.5.4, sodass wir diese anwenden können und damit die gewünschten Resultate erhalten. □

Hat man also z. B. eine Fehler-Funktion von $\varepsilon = \frac{1}{\log n}$, so kann das parametrisierte Clique-Problem in FPT-Zeit gelöst werden (und sogar in Linearzeit, wenn k klein genug ist, d. h., $k < k'$ mit der in Satz 4.5.3 implizit gegebenen Funktion k').

Der angegebene beschränkte Suchbaum-Algorithmus 4.4 besitzt gute Laufzeiten, wenn der Parameter k klein ist. Für Werte für k, die größer als die erwartete Größe einer Maximum-Clique sind, ist damit nur eine superpolynomielle (aber noch subexponentielle) erwartete Laufzeit zu erhalten. Durch eine Reduktion auf das Vertex-Cover-Problem kann man jedoch bessere Resultate erhalten:

Satz 4.5.6 (Smoothed-Komplexität für das Clique-Problem mit konstanter Stör-Wahrscheinlichkeit)**.** *Seien $n \in \mathbb{N}$ und $G = (V, E)$ ein Graph mit $n := |V|$ Knoten. Weiterhin seien $0 < \varepsilon \leq \frac{1}{2}$, $0 < \tilde{\varepsilon}$ Konstanten und $k > \tilde{\varepsilon} n$. Dann besitzt das* Clique*-Problem eine polynomielle parametrisierte Smoothed-Komplexität $SC(n, k, \varepsilon)$.*

Beweis. Das Finden einer (Maximum-) Clique in einem (p, q)-beschränkten Zufallsgraph ist äquivalent zum Finden einer (Maximum-) unabhängigen Menge im inversen $(1 - q, 1 - p)$-beschränkten Zufallsgraph. Damit sind die Probleme äquivalent. Diese Reduktion erhält die Zugehörigkeit zu avg−FPT, da sich weder Parameter noch Problemgröße ändern, während sich der zugrunde liegende Wahrscheinlichkeitsraum in der gleichen Art verändert wie der Graph selbst (Übergang zum Inversen), sodass der ursprüngliche Graph im ursprünglichen Wahrscheinlichkeitsraum und der Bildgraph im Bild-Wahrscheinlichkeitsraum mit identischer Wahrscheinlichkeit gezogen werden. Abschließend ist diese Reduktion in $\mathcal{O}(n)$ und damit (avg−)P-Zeit berechenbar. Also sind das Clique- und das Independent-Set-Problem äquivalent im Sinne einer (avg−)P-Reduktion.

Aber auch das Independent-Set- und das Vertex-Cover-Problem sind eng miteinander verwandt: Ein Graph mit n Knoten besitzt eine (Minimum-) Knotenüberdeckung der Größe k genau dann, wenn er eine (Maximum-) unabhängige Menge der Größe $n - k$ besitzt: Alle Knoten, die nicht an einer gegebenen Knotenüberdeckung beteiligt sind, bilden eine unabhängige Menge, und umgekehrt.

Zusammen ergibt sich: Unter einem zusätzlichen Zeitaufwand von $\mathcal{O}(n)$ kann man das Clique- auf das Vertex-Cover-Problem reduzieren. Damit kann Satz 4.4.8 zur Anwendung kommen und erhält die dortigen Resultate. Da diese den hier angegebenen entsprechen, ist damit der Beweis vollbracht. □

4.6 Das (d)-Hitting-Set-Problem

Beim (parametrisierten) Hitting-Set-Problem ist eine Grundmenge G von n Elementen sowie eine Mengenfamilie $\mathcal{F}$ von Teilmengen von G und ein Parameter k gegeben. Gesucht ist eine Teilmenge S von G der Größe k, sodass jedes Element $X \in \mathcal{F}$ einen nicht-leeren Schnitt mit S besitzt. Gilt zusätzlich für alle Elemente $X \in \mathcal{F}$, dass sie jeweils genau d-elementig sind, erhält man das d-Hitting-Set-Problem.

Man bemerke, dass manche Autoren mit der Variablen n in diesem Kontext nicht die Mächtigkeit der Grundmenge G, sondern die der Mengenfamilie $\mathcal{F}$ bezeichnen.

Um diese beiden Probleme zu lösen, verwenden wir Algorithmus 4.5. Dieser ist korrekt, da eine leere Menge mit keiner einen nicht-leeren Schnitt besitzt, d. h., wenn sich eine solche in der Berechnung als Element der Mengenfamilie $\mathcal{F}$ ergibt, gibt es keine Lösung. Und die Elemente einelementiger Teilmengen müssen notwendigerweise auch Teil der Lösungsmenge sein.

Wir zeigen nun zwei Lemmata, die eine allgemeine Einsicht über das Verhalten des Algorithmus geben.

Lemma 4.6.1. *Algorithmus 4.5 verallgemeinert Algorithmus 4.3.*

Beweis. Es ist ein Element bzw. Knoten $v \in G$ Teil der Lösungsmenge, oder alle seine Nachbarn (d. h. Elemente w mit $\{v, w\} \in \mathcal{F}$). Letzteres gilt, da nach dem Löschen von v aus all diesen zwei-elementigen Mengen $\{v, w\}$ nur noch die einelementigen $\{w\}$ übrigbleiben, sodass jedes solche Element w Teil der Lösungsmenge sein muss. □

Lemma 4.6.2. *Folgt für $n > d$ die Eingabe-Mengenfamilie $\mathcal{F}$ von Algorithmus 4.5 der Wahrscheinlichkeitsverteilung $RS(n, p, q)$ bzw. $RS_d(n, p)$, so sind die in den Rekursionsaufrufen betrachteten Mengenfamilien gemäß der Wahrscheinlichkeitsverteilungen $RS(n-1, p, q)$ bzw. $RS_d(n-1, p)$ verteilt.*

Beweis. Für die Rekursionsaufrufe gilt, dass, wenn $\mathcal{F} \sim RS(n, p, q)$, also alle Einzelwahrscheinlichkeiten durch p und q beschränkt sind, dies auch (bei nun um ein Element kleinerer Grundmenge) für die Mengenfamilien in den Rekursionsaufrufen der Fall ist. Der Zufallsraum bleibt also „erhalten“.

Genauso sind für $\mathcal{F} \sim RS_d(n,p)$ nach dem Löschen des Elements v die Mengenfamilien in den Rekursionsaufrufen gemäß $RS_d(n-1,p)$ verteilt: Es sind nur noch diejenigen d-elementigen Teilmengen in der neuen Mengenfamilie enthalten, die nicht das gelöschte Element v enthalten. Diese waren aber auch schon zuvor p-beschränkt. Hinzu kommen höchstens noch Mengen, die durch das Löschen von v aus einer zuvor $d+1$-elementigen Menge entstanden sind. Dies erhöht aber nur für die nun noch betrachteten d-elementigen Mengen deren Auftrittswahrscheinlichkeiten, verringert sie aber nicht, sodass die p-Beschränktheit weiterhin erhalten bleibt. □

Algorithmus 4.5 : Einfacher beschränkter Suchbaum-Algorithmus für das HITING SET-Problem

Input : Familie $\mathcal{F}$ von Teilmengen einer Grundmenge G und Parameter k
Output : Existiert eine Lösungsmenge $S \subset G$ mit $|S| = k$, sodass jedes Element von $\mathcal{F}$ nichtleeren Schnitt mit S hat? (Wenn „ja“, gib eine solche Lösung zurück.)

2 **if** *Ist die leere Menge oder sind mehr als k einelementige Mengen Element von $\mathcal{F}$?* **then**
3 **return** *„nein“.*
4 Wähle eine Menge $X \in \mathcal{F}$ mit minimaler Elementanzahl und daraus ein Element $v \in X$.
 `// Nimm zuerst an, dass` v `Teil der Lösung ist. Löse das verbleibende Teilproblem rekursiv.`
5 Lösche dafür v aus G und jede Teilmenge von G in $\mathcal{F}$, welche v enthielt. Löse das Problem rekursiv auf dem induzierten Teilproblem mit zugehörigem Parameter $k-1$.
6 **if** *Wurde eine Lösung gefunden?* **then**
7 **return** *„ja“ und die rekursiv gefundene Lösungsmenge vereinigt mit $\{v\}$ selbst.*
8 **else** `// Wurde keine Lösung gefunden, in der` v `enthalten ist, suche nach einer ohne dieses Element.`
9 Lösche v aus G sowie allen Mengen in $\mathcal{F}$ und löse das Problem rekursiv auf diesem induzierten Teilproblem.
10 **if** *Wurde eine Lösung gefunden?* **then return** *„ja“ und die entsprechende Lösung.*
11 **return** *„nein“.*

Damit können wir bei der Analyse von Algorithmus 4.5 ähnlich vorgehen wie bei Algorithmus 4.3, denn genau wie dort, „vererbt“ sich der Zufallsraum in die Rekursionsaufrufe, sodass wir die entstehende Rekursionsgleichung bei der Betrachtung der erwarteten Laufzeit entsprechend lösen können.

Nun soll die Eigenschaft von Algorithmus 4.5 betrachtet werden, dass der Suchbaum sich nicht zu sehr verzweigen kann. Wir zeigen dafür, dass mit hoher Wahrscheinlichkeit nur eine beschränkte Anzahl an aufeinanderfolgenden Löschungen von Elementen der Grundmenge stattfindet.

Dazu benötigen wir zuerst ein kombinatorisches Lemma, welches eine einfache Abschätzung für die Größe von Binomialkoeffizienten angibt.

Lemma 4.6.3. *Es seien s und d positive ganze Zahlen mit $s \geq \ell \cdot d$ für eine reelle Zahl $\ell \geq 1$. Dann ist*

$$\binom{s}{d} \geq \ell^d.$$

Beweis. Es ist

$$\binom{s}{d} \geq \ell^d = \prod_{i=0}^{d-1} \frac{s-i}{d-i} \geq \prod_{i=0}^{d-1} \frac{\ell \cdot d - i}{d-i} \geq \prod_{i=0}^{d-1} \frac{\ell \cdot (d-i)}{d-i} = \ell^d.$$

□

Jetzt können wir eine Aussage darüber treffen, wie viele Elemente höchstens ausgeschlossen werden können, bevor der Algorithmus mit hoher Wahrscheinlichkeit terminiert.

Lemma 4.6.4. *Sei n eine natürliche Zahl und $k, d : \mathbb{N} \to \mathbb{N}$ mit $k = o(n)$ und $2 \leq d = o(n)$. Weiterhin sei $p\colon \mathbb{N} \to [0,1]$ eine Wahrscheinlichkeitsfunktion mit $p = \omega\left(\left(\frac{d}{n}\right)^{d-1}\right)$. Darüber hinaus sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F} \sim RS_d(n,p)$ eine Mengenfamilie von Teilmengen von G. Dann existiert eine Konstante $c > 0$, dass Algorithmus 4.5 mit Wahrscheinlichkeit $1 - 2^{-n}$ nach spätestens $c \cdot d \cdot \sqrt[d-1]{\frac{1}{p}}$ erfolgten Löschungen von Elementen, ohne diese in die Lösungsmenge aufzunehmen, eine Mengenfamilie mit mehr als k einelementigen Mengen betrachtet.*

Beweis. Es sei S die Menge der vorher auf diesem Berechnungspfad im Suchbaum gemäß Schritt 5 von Algorithmus 4.5 ausgeschlossenen Elemente und $s := |S|$. Weiterhin sei v ein bisher noch nicht betrachtetes Element der Grundmenge. Die einelementige Menge $\{v\}$ existiert nun mindestens dann in der Mengenfamilie des aktuellen Rekursionsaufrufs, wenn es eine $(d-1)$-elementige Teilmenge M von S gibt, sodass $M \cup \{v\}$ in der ursprünglichen Mengenfamilie $\mathcal{F}$ enthalten war. Dies war nach Voraussetzung mit Wahrscheinlichkeit von mindestens p der Fall, d. h., mit Wahrscheinlichkeit höchstens $1-p$ war diese konkrete Menge nicht in $\mathcal{F}$ vorhanden.

Nun gibt es $\binom{s}{d-1}$ solcher $(d-1)$-elementiger Teilmengen M von S. Und für jede davon lässt sich ein eigenes, d-elementiges, potentielles Element $M \cup \{v\}$ von $\mathcal{F}$ konstruieren. Da jedes dieser unabhängig voneinander mit Wahrscheinlichkeit höchstens $1 - p < \exp(-p)$ in $\mathcal{F}$ nicht vertreten ist, kommt dort nur mit einer Wahrscheinlichkeit von weniger als $P(s) := \exp\left(-\binom{s}{d-1}p\right)$ keiner dieser Mengen vor. Damit ist also auch gleichzeitig die Wahrscheinlichkeit, dass $\{v\}$ als einelementige Menge in der im aktuellen Rekursionsaufruf betrachteten Mengenfamilie nicht enthalten ist, nach oben durch diesen Wert $P(s)$ beschränkt.

Setzen wir also $s := c \cdot d \cdot \sqrt[d-1]{\frac{1}{p}}$ für eine noch zu bestimmende, positive Konstante c, so gilt demnach $s \geq \left(c \cdot \sqrt[d-1]{\frac{1}{p}}\right) \cdot (d-1)$ und damit nach Lemma 4.6.3

$$\binom{s}{d-1} \cdot p \geq c^{d-1} \cdot \frac{1}{p} \cdot p = c^{d-1}.$$

Schließlich ist

$$P(s) = \exp\left(-\binom{s}{d-1}p\right) < \exp\left(-c^{d-1}\right).$$

Wählen wir $c := -\left(\log c_1\right)^{\frac{1}{d-1}}$ für eine noch zu bestimmende Konstante $0 < c_1 < 1$, so erhalten wir $P(s) < c_1$ für diesen Wert von s, den wir ja direkt von c_1 abhängig definiert haben. Anders formuliert: Nach s ausgeschlossenen Elementen ist für jedes weitere noch mögliche Element v die entsprechende einelementige Menge mit Wahrscheinlichkeit weniger als c_1 in der nun betrachteten Mengenfamilie nicht enthalten. Da im Berechnungsprozess von Algorithmus 4.5 vom Start- bis zum aktuell betrachteten Rekursionsaufruf nur nacheinander die Elemente aus S sowie die dabei schon als Teil der Lösungsmenge ausgewählten höchstens $k = o(n)$ Elemente betrachtet wurden, stehen noch weitere $n - s - o(n)$ solcher Elemente v zur Verfügung. Da aber

$$s = c \cdot d \cdot \sqrt[d-1]{\frac{1}{p}} = o\left(c \cdot d \cdot \frac{n}{d}\right) = o(n)$$

ist, wurde bisher nur eine Nullmenge von Elementen der Grundmenge G betrachtet und es stehen weitere $n - s - o(n) = n \cdot (1 - o(1))$ zur Auswahl. Von diesen sind also im Erwartungswert nur von weniger als $\mu := (c_1 - o(1)) \cdot n$ Elementen aus G die entsprechenden einelementigen Mengen nicht in der im aktuellen Rekursionsaufruf von Algorithmus 4.5 betrachteten Mengenfamilie enthalten. Damit aber höchstens $k = o(n)$ solche einelementigen Mengen vorkommen, dürfen die restlichen $(1-o(1))\cdot n$ eben nicht enthalten sein. Diese Eigenschaft muss also für mehr als die $\frac{1-o(1)}{c_1-o(1)} > \frac{1}{c_1} - o(1) =: 1 + \delta$ -fache Anzahl von Elementen im Vergleich zur erwarteten Anzahl gelten. Dies tritt aber nur ein mit einer durch die Chernoff-Schranke in Lemma 2.8.1 bestimmten Wahrscheinlichkeit (und alle genügend großen n) von weniger als

$$\begin{aligned}
\exp\left(-\frac{\delta^2}{3}\mu\right) &= \exp\left(-\frac{1}{3} \cdot \left(\frac{1}{c_1} - 1 - o(1)\right)^2 \cdot (c_1 - o(1)) \cdot n\right) \\
&= \exp\left(-\frac{1}{3} \cdot \left(\frac{1}{c_1} - o(1) - 2 - o(1) + c_1 + o(1)\right) \cdot n\right) \\
&< \exp\left(-\frac{1}{3}\left(\frac{1}{c_1} - 3\right) \cdot n\right) = \exp\left(-\left(\frac{1}{3c_1} - 1\right) \cdot n\right).
\end{aligned}$$

Wählt man nun c_1 so, dass $\left(\frac{1}{3c_1} - 1\right)$ größer als $\log 2$ wird, fällt diese Wahrscheinlichkeit also auf weniger als 2^{-n}, d. h., nur mit dieser Wahrscheinlichkeit sind dann weniger als k einelementige Mengen vorhanden.

Anders formuliert: Damit sind mit einer Wahrscheinlichkeit von $1 - 2^{-n}$ nach einer Anzahl s von Ausschlüssen mehr als k einelementige Mengen in der betrachteten Mengenfamilie vorhanden; und der Beweis ist dadurch abgeschlossen. □

Dieses Lemma gilt für alle Wahrscheinlichkeiten p. Eine deutliche Verschärfung für „große“ p erhält man, wenn man zuerst den Parameter d „verkleinert“:

Lemma 4.6.5. *Sei n eine natürliche Zahl und $k, d : \mathbb{N} \to \mathbb{N}$ mit $k = o(n)$ und $2 \leq d = o(n)$. Weiterhin sei $p\colon \mathbb{N} \to [0,1]$ eine Wahrscheinlichkeitsfunktion mit $\exp(-d) < p \leq \exp(-(1+\varepsilon))$ für eine Konstante $\varepsilon > 0$. Darüber hinaus sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F} \sim RS_d(n,p)$ eine Mengenfamilie von Teilmengen von G. Dann existiert eine Konstante $c > 0$, dass Algorithmus 4.5 mit Wahrscheinlichkeit $1 - 2^{-n}$ nach spätestens $d + c \cdot \log\left(\frac{1}{p}\right)$ erfolgten Löschungen von Elementen, ohne diese in die Lösungsmenge aufzunehmen, eine Mengenfamilie mit mehr als k einelementigen Mengen betrachtet.*

Beweis. Wir betrachten zuerst das im Schritt 4 von Algorithmus 4.5 betrachtete Element v und die Familie $\mathcal{F}_d(v)$ aller d-elementigen Teilmengen der Grundmenge G, die v enthalten. Nach Voraussetzung ist die Auftrittswahrscheinlichkeit einer beliebigen Menge aus $\mathcal{F}_d(v)$ nach unten durch p beschränkt. Löscht man nun gemäß Schritt 9 aus allen diesen Mengen das Element v, erhält man alle $(d-1)$-elementigen Teilmengen von G, sodass dann deren Auftrittswahrscheinlichkeiten jeweils zumindest p betragen. Die nun betrachtete Mengenfamilie gehört also nicht nur der Klasse $RS_d(n-1,p)$, sondern auch $RS_{d-1}(n-1,p)$ an. Analog erhält man damit nach dem Löschen von $d - d'$ Elementen eine Mengenfamilie der Klasse $RS_{d'}(n-d',p)$.

Wir wollen nun mit $d' = \log\left(\frac{1}{p}\right)$ das Lemma 4.6.4 anwenden. Diese Wahl für d' ist wegen $\frac{1}{p} < \exp(d)$ und somit $d' < d$ erst einmal möglich. Es verbleibt noch zu prüfen, ob die Voraussetzung

$$p = \omega\left(\left(\frac{d'}{n}\right)^{d'-1}\right)$$

erfüllt ist. Setzen wir d' ein und kürzen $\frac{1}{p} > \exp(1+\varepsilon) =: \tilde{p}$ ab, so ist dies äquivalent zu

$$\tilde{p} = o\left(\left(\frac{n}{\log \tilde{p}}\right)^{\log \tilde{p}-1}\right).$$

Zieht man nun die entsprechende Wurzel, erhält man die weitere Äquivalenz mit

$$\tilde{p}^{\frac{1}{\log \tilde{p}-1}} = o\left(\frac{n}{\log \tilde{p}}\right).$$

Dabei ist die linke Seite wegen

$$\tilde{p}^{\frac{1}{\log \tilde{p}-1}} = \exp\left(\log \tilde{p} \cdot \frac{1}{\log \tilde{p}-1}\right) = \exp\left(1 + \frac{1}{\log \tilde{p}-1}\right) \leq \exp\left(1 + \frac{1}{\varepsilon}\right)$$

nach oben beschränkt. Also genügt es dann, wenn $\frac{n}{\log \tilde{p}}$ unbeschränkt wächst, d. h. $\log \tilde{p} = o(n)$ bzw. $p = \exp(-o(n))$ ist, was nach Voraussetzung wegen $d = o(n)$ und $p > \exp(-d)$ auch gilt.

Daher können wir Lemma 4.6.4 auf die aktuelle Situation mit d' statt d anwenden und erhalten nach weiteren höchstens

$$c \cdot d' \cdot \sqrt[d'-1]{\frac{1}{p}} = c \cdot \log \tilde{p} \cdot \tilde{p}^{\frac{1}{\log \tilde{p}-1}} < \tilde{c} \cdot \log \tilde{p}$$

ausgeschlossenen Elementen eine Situation, in der mit Wahrscheinlichkeit $1 - 2^{-n}$ mehr als k einelementige Mengen vorhanden sind. Insgesamt wurden dafür dann höchstens $d - d' + \tilde{c} \cdot \log \tilde{p} < d + \tilde{c} \cdot \log \frac{1}{p}$ Ausschlüsse benötigt, was zu beweisen war. □

Die obere Schranke an die Wahrscheinlichkeit p ist dabei keine Einschränkung an die individuellen Auftrittswahrscheinlichkeiten der einzelnen Mengen. Gefordert war ja nur, dass diese jeweils immer nur mindestens p betragen, aber natürlich auch alle echt größer sein können. Insofern folgt schnell das

Korollar 4.6.6. *Ist in der Situation von Lemma 4.6.5 die Wahrscheinlichkeitsfunktion $p > \exp(-(1+\varepsilon)$ für eine Konstante $\varepsilon > 0$, so gibt es eine Konstante $c > 0$, dass nach spätestens $d + c \cdot (1 + \varepsilon)$ ausgeschlossenen Elementen mit Wahrscheinlichkeit von mindestens $1 - 2^{-n}$ mehr als k einelementige Mengen vorhanden sind.*

Beweis. Wählt man $\tilde{p} := \exp((-(1 + \varepsilon))$ und wendet darauf das Lemma 4.6.5 an, erhält man für die notwendige Anzahl ausgeschlossener Elemente einen Wert von höchstens $d + c \cdot \log \frac{1}{\tilde{p}} = d + c \cdot (1 + \varepsilon)$. Da aber $p > \tilde{p}$ ist, steigen beim Übergang von $\tilde{p}$ zu p nur die Auftrittswahrscheinlichkeiten der einzelnen Mengen, was also auch für die betrachteten einelementigen Mengen gilt und so die gewünschte Aussage zeigt. □

Um den ersten Satz über Algorithmus 4.5 zum d-HITTING-SET-Problem formulieren zu können, benötigen wir zuletzt noch eine geeignete Definition der Laufzeit des Algorithmus:

Definition 4.6.7. Seien n, d, k und s natürliche Zahlen, wobei alle jeweils höchstens den Wert n besitzen. Weiterhin sei $p\colon \mathbb{N} \to [0,1]$ eine Wahrscheinlichkeitsfunktion und $\mathcal{F} \sim RS_d(n,p)$ eine Mengenfamilie von Teilmengen einer Grundmenge mit n Elementen. Dann sei $T_{p,d}(n,k)$ die erwartete Laufzeit von Algorithmus 4.5 für die Eingabe von $\mathcal{F}$ und k sowie $T_{p,d}(n,k,s)$ die entsprechende Laufzeit, wenn bei der Berechnung schon zuvor s Elemente als nicht Teil der Lösungsmenge ausgeschlossen wurden. Betrachten wir Mengenfamilien $\mathcal{F}$ mit $\mathcal{F} \sim RS(n,p)$, so werden die erwarteten Laufzeiten analog definiert und der Index d weggelassen.

Insbesondere ist also $T_{p,d}(n,k) = T_{p,d}(n,k,0)$.

Nun können wir die Laufzeit des Algorithmus bestimmen:

Satz 4.6.8 (Das d-Hitting-Set-Problem ist in avg−FPT)**.** *Sei n eine natürliche Zahl und $k, d : \mathbb{N} \to \mathbb{N}$ mit $k = o(n)$ und $2 \leq d = o(n)$. Weiterhin sei $p\colon \mathbb{N} \to [0,1]$ eine Wahrscheinlichkeitsfunktion mit $\exp(-d) < p \leq \exp(-(1+\varepsilon))$ für eine Konstante $\varepsilon > 0$. Darüber hinaus sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F} \sim RS_d(n,p)$ eine Mengenfamilie von Teilmengen von G. Dann existiert eine Konstante $c > 0$, dass Algorithmus 4.5 eine erwartete Laufzeit von*

$$T_{p,d}(n,k) = \binom{k + d + c \cdot \log \frac{1}{p}}{k} \cdot \mathcal{O}\left(\left(k + d + c \cdot \log \frac{1}{p}\right)|\mathcal{F}|\right)$$

besitzt. Insbesondere ist also das d-Hitting-Set-Problem in avg−FPT.

Beweis. Wir betrachten zuerst nur die Anzahl der Rekursionsaufrufe und ignorieren die dabei jeweils benötigten Anpassungen und Berechnungen der Mengen in der Mengenfamilie $\mathcal{F}$. Mit der Rekursion des Algorithmus 4.5 erhalten wir damit die vereinfachte Rekursionsgleichung

$$T_{p,d}(n,k,s) = T_{p,d}(n-1,k-1,s) + T_{p,d}(n-1,k,s+1), \tag{4.11}$$

für $k > 0$ und $s < d + c \cdot \log \frac{1}{p}$,

$$T_{p,d}(n,k,d + c \cdot \log \frac{1}{p}) = \mathcal{O}(1) \text{ sowie} \tag{4.12}$$

$$T_{p,d}(n,0,s) = \mathcal{O}(1). \tag{4.13}$$

Während sich Gleichung (4.11) direkt aus den beiden Rekursionsaufrufen in den Schritten 5 und 9 sowie Gleichung (4.13) aus Schritt 2 von Algorithmus 4.5 ergeben, benötigt Gleichung (4.12) eine ausführlichere Begründung:

Sind schon $d+c\cdot\log\frac{1}{p}$ Elemente auf dem bisherigen Berechnungspfad ausgeschlossen worden, so existieren nach Lemma 4.6.5 mit einer Wahrscheinlichkeit von mindestens $1-2^{-n}$ nun mehr als k einelementige Mengen in der nun betrachteten Mengenfamilie, sodass mit dem Schritt 2 wieder in $\mathcal{O}(1)$ die Berechnung abgeschlossen werden kann. Das Gegenereignis, dass die Berechnung also fortgesetzt werden muss, tritt nur mit einer Wahrscheinlichkeit von weniger als 2^{-n} ein. Da aber höchstens 2^n Rekursionsaufrufe erfolgen können (da in jedem Schritt höchstens zwei Aufrufe mit einer Grundmenge der Größe $n-1$ erfolgen), trägt auch dieser Fall im Erwartungswert nur mit $\mathcal{O}(1)$ Rekursionsaufrufen bei.

Damit ergibt sich also folgendes Bild: Bei jedem Rekursionsaufruf wird entweder der Parameter k (bis 0) herunter-, oder der Parameter s (bis $d+c\cdot\log\frac{1}{p}$) heraufgezählt. Dies geschieht so lang, bis mindestens einer der beiden Parameter an seine Grenze stößt. Da auf einem Berechnungspfad diese „k-“ bzw. „s-Schritte“ in beliebiger Reihenfolge erfolgen können, ist deren Gesamtanzahl durch die Anzahl aller Permutationen von k solcher „k-“ und $d+c\cdot\log\frac{1}{p}$ solcher „s-Schritte“ beschränkt. Aufgrund der Wiederholungen gibt es davon genau $\binom{k+d+c\cdot\log\frac{1}{p}}{k}$ Stück. Dies ist also eine Abschätzung der Anzahl aller Berechnungspfade.

Auf jedem dieser Berechnungspfade wird nun bei jedem Rekursionsaufruf jede Menge in der Mengenfamilie höchstens konstant oft betrachtet. Da weder bei der Löschung von Elementen aus den Mengen noch von Mengen selbst aus der Mengenfamilie die Anzahl der Mengen in der Mengenfamilie erhöht wird, ist diese Anzahl an Operationen immer in $\mathcal{O}(\mathcal{F})$.

Für jeden einzelnen Berechnungspfad finden $k+d+c\cdot\log\frac{1}{p}$ Rekursionsaufrufe statt, bis die Parameter k und s ihre jeweilige Grenze erreicht haben. Danach folgen mit einer Wahrscheinlichkeit von weniger als 2^{-n} noch weitere n Rekursionsaufrufe, d. h. im Erwartungswert $o(1)$ weitere. Insgesamt finden also auf jedem solchen Berechnungspfad $\mathcal{O}\left(\left(k+d+c\cdot\log\frac{1}{p}\right)|\mathcal{F}|\right)$ einzelne Mengenoperationen statt. Und da es davon höchstens $\binom{k+d+c\cdot\log\frac{1}{p}}{k}$ verschiedene gibt, erhalten wir als Produkt die postulierte erwartete Laufzeit.

Da diese die Form $f(k,d)\cdot|\mathcal{F}|$ besitzt und die Eingabelänge durch $|\mathcal{F}|$ gegeben ist, besitzt der Algorithmus 4.5 – und damit das d-Hitting-Set-Problem – eine erwartete FPT-Laufzeit, d. h., das Problem ist in avg−FPT. □

Um das Laufzeitverhalten besser mit den bekannten Ergebnissen der Worst-Case-Analyse vergleichen zu können, geben wir eine andere Abschätzung an, die zumindest für kleine Werte von d ein ähnlich gutes Resultat liefert:

Korollar 4.6.9. *Sind in der Situation von Satz 4.6.8 die Wahrscheinlichkeitsfunktion $p\leq\exp(-(1+\varepsilon))$ und der Parameter d konstant mit $d\leq k$, so existiert eine von p abhängige Konstante $c>0$, sodass die erwartete Laufzeit $T_{p,d}(n,k)$ von Algorithmus 4.5 in $\mathcal{O}\left(k^{d+c}\cdot|\mathcal{F}|\right)$ liegt.*

Beweis. Für konstante p ist auch $c \cdot \log \frac{1}{p}$ konstant und der Binominialkoeffizient aus dem Ergebnis von Satz 4.6.8 kann durch die Ungleichung $(k+d+\tilde{c})^{d+\tilde{c}} < k^{d+\bar{c}}$ nach oben abgeschätzt werden. □

Diese nur polynomielle Abhängigkeit vom Parameter k ist damit deutlich besser als die bisher besten bekannten Abschätzungen. Diese besitzen nämlich im Worst-Case-Fall mit $\left(\frac{d-1}{2}\left(1+\sqrt{1+\frac{4}{(d-1)^2}}\right)\right)^k$, siehe [41] (wobei dort auch für kleine Werte von d kleinere Basen in der exponentiellen Abhängigkeit gezeigt werden konnten), für allgemeine Werte von d, aber auch für den Spezialfall $d = 3$ mit $2,0755^k$, siehe [97], nur exponentielle Abhängigkeiten vom Parameter k.

Dass das d-Hitting-Set-Problem bei einer Average-Case-Betrachtung eine FPT-Laufzeit zur Lösung besitzt, ist nicht weiter verwunderlich, da dies auch schon bei einer Worst-Case-Betrachtung der Fall ist. Jedoch können wir ein ähnliches Ergebnis auch für das allgemeinere Hitting-Set-Problem zeigen, welches im Worst-Case sogar W[2]-vollständig ist:

Satz 4.6.10 (Das Hitting-Set-Problem ist in avg−FPT)**.** *Seien n und k natürliche Zahlen mit $k = o(n)$. Weiterhin sei $p\colon \mathbb{N} \to [0,1]$ eine Wahrscheinlichkeitsfunktion mit $p \leq \exp(-(1+\varepsilon))$ für eine Konstante $\varepsilon > 0$. Darüber hinaus sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F} \sim RS(n,p)$ eine Mengenfamilie von Teilmengen von G. Dann existiert eine Konstante $c > 0$, sodass Algorithmus 4.5 eine erwartete Laufzeit von*

$$T_{p,d}(n,k) = \binom{k + c \cdot \log \frac{1}{p}}{k} \cdot \mathcal{O}\left(\left(k + c \cdot \log \frac{1}{p}\right)|\mathcal{F}|\right)$$

besitzt. Insbesondere ist also das Hitting-Set-*Problem in* avg−FPT*.*

Beweis. Da die Auftritts-Wahrscheinlichkeiten aller Teilmengen der Grundmenge durch p nach unten beschränkt sind, gilt dies also auch für alle d-elementigen, für ein geeignetes, noch zu wählendes d. Insbesondere ist also auch $\mathcal{F} \sim RS_d(n,p)$. Setzen wir $d := \log \frac{1}{p}$, so ist $p = \exp(-d)$ und wir können Satz 4.6.8 anwenden. Das liefert das gewünschte Laufzeit-Ergebnis sowie die Zugehörigkeit des Problems zur Komplexitätsklasse avg−FPT. □

Auch hier wollen wir eine Vereinfachung für konstante Wahrscheinlichkeiten $p > 0$ angeben:

Korollar 4.6.11. *Ist in der Situation von Satz 4.6.10 der Wert $p > 0$ konstant, dann existiert eine von p abhängige Konstante $c > 0$, sodass die erwartete Laufzeit $T_p(n,k)$ von Algorithmus 4.5 in $\mathcal{O}\left(k^c \cdot |\mathcal{F}|\right)$ liegt.*

Beweis. Für $p \leq \exp(-(1+\varepsilon))$ liefert Einsetzen und Vereinfachen des Resultats von Satz 4.6.10 direkt das gewünschte Ergebnis. Da die Laufzeit monoton fallend in p ist, gilt die angegebene Abschätzung dann auch für entsprechend größere untere Schranken p an die individuellen Auftrittswahrscheinlichkeiten der einzelnen Mengen. □

Schließlich können wir diese Erkenntnisse durch das gewählte Zufallsmengen-Modell auf eine Smoothed-Betrachtung übertragen:

Satz 4.6.12 (Smoothed-Komplexität von d-Hitting-Set)**.** *Seien n und d natürliche Zahlen und $k : \mathbb{N} \to \mathbb{N}$ mit $d \leq k = o(n)$. Weiterhin sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F}$ eine Mengenfamilie von d-elementigen Teilmengen von G sowie $0 < \varepsilon \leq \frac{1}{2}$ eine konstante Stör-Wahrscheinlichkeit, wobei nur die d-elementigen Teilmengen von G in ihrer Auftrittswahrscheinlichkeit gestört werden. Dann existiert eine Konstante $c = c(\varepsilon) > 0$, dass Algorithmus 4.5 eine parametrisierte Smoothed-Komplexität von*

$$SC(n, k, d, \varepsilon) = \mathcal{O}\left(k^{d+c} \cdot n^d\right)$$

besitzt.

Beweis. Durch die Störung erhält man eine Mengenfamilie $\tilde{\mathcal{F}}$, die dem Wahrscheinlichkeitsraum $RS_d(n, \varepsilon)$ angehört. Ist $\varepsilon < \frac{1}{e}$, kann man nun direkt das Korollar 4.6.9 anwenden. Berücksichtigt man noch, dass $\tilde{\mathcal{F}}$ höchstens die $\binom{n}{d} = \mathcal{O}\left(n^d\right)$ d-elementigen Teilmengen von G als Elemente enthält, ergibt sich die angegebene erwartete Laufzeit. Da das Laufzeitverhalten monoton fallend in der unteren Schranke an die Auftrittswahrscheinlichkeiten der einzelnen Mengen ist, gilt die Aussage aber auch für alle größeren Werte von ε. □

Wendet man anstatt Korollar 4.6.9 den Satz 4.6.8 an, erhält man eine allgemeinere Version des eben gezeigten Satzes, die auch gegen 0 gehende Stör-Wahrscheinlichkeiten abdeckt. Analog erhalten wir auf diese Weise für das allgemeinere Hitting-Set-Problem:

Satz 4.6.13 (Smoothed-Komplexität von Hitting-Set)**.** *Sei n eine natürliche Zahl und $k : \mathbb{N} \to \mathbb{N}$ mit $k = o(n)$. Weiterhin sei G eine Grundmenge mit $|G| = n$ und $\mathcal{F}$ eine Mengenfamilie von Teilmengen von G. Schließlich sei $\varepsilon \colon \mathbb{N} \to [0, 1]$ eine Stör-Wahrscheinlichkeit mit $\varepsilon < \frac{1}{e}$. Werden nur die Teilmengen mit höchstens $\log \frac{1}{\varepsilon}$ Elementen gestört, dann existiert eine Konstante $c > 0$, dass Algorithmus 4.5 eine parametrisierte Smoothed-Komplexität von*

$$SC(n, k, \varepsilon) = \binom{k + c \cdot \log \frac{1}{\varepsilon}}{k} \cdot \mathcal{O}\left(\left(k + c \cdot \log \frac{1}{\varepsilon}\right)\left(n^{\log \frac{1}{\varepsilon}} + |\mathcal{F}|\right)\right)$$

besitzt.

Dabei kann zusätzlich angenommen werden, dass auch alle Teilmengen der Grundmenge mit höchstens $\log \frac{1}{\varepsilon} - 1$ Elementen nicht gestört werden. (Insbesondere erhält man dadurch also keine zusätzliche sehr kleine Mengen in der verrauschten Mengenfamilie.)

Beweis. Analog zum Beweis von Satz 4.6.12 erhalten wir durch das Verrauschen eine Mengenfamilie $\tilde{\mathcal{F}}$, die dem Wahrscheinlichkeitsraum $RS_d(n, \varepsilon)$ für $d = \log \frac{1}{\varepsilon}$ angehört. Damit kann Satz 4.6.8 angewendet werden. Berücksichtigt man, dass die verrauschte Mengenfamilie $\tilde{\mathcal{F}}$ neben den Elementen aus $\mathcal{F}$ höchstens noch alle d-elementigen Teilmengen von G enthält, von denen es $n^d = n^{\log \frac{1}{\varepsilon}}$ gibt, so erhält man das angegebene Ergebnis. □

Wieder kann man aufgrund der Monotonie Abschätzungen für die Smoothed-Komplexität für Werte von $\frac{1}{e} \leq \varepsilon \leq \frac{1}{2}$ erhalten, indem man in den Satz ein $\tilde{\varepsilon} < \frac{1}{e}$ einsetzt und somit eine obere Schranke für den gesuchten Wert erhält.

Abschließend wollen wir auch hier das Resultat für konstante Stör-Wahrscheinlichkeiten ε vereinfachen:

Korollar 4.6.14. *Ist in der Situation von Satz 4.6.13 die Stör-Wahrscheinlichkeit $0 < \varepsilon \leq \frac{1}{2}$ konstant, so existiert eine Konstante $c = c(\varepsilon) > 0$, sodass die parametrisierte Smoothed-Komplexität von Algorithmus 4.5 gegeben ist durch*

$$SC(n, k, \varepsilon) = k^c \cdot \mathcal{O}\left(n^c + |\mathcal{F}|\right).$$

Beweis. Für $\varepsilon < \frac{1}{e}$ ergibt sich dies direkt durch Einsetzen in die Aussage von Satz 4.6.13 und nachträgliches Abschätzen; für die größeren Werte durch die Monotonie. □

4.7 Zusammenfassung

Wir haben das Average-Case- und Smoothed-Verhalten dreier archetypischer FPT-, W[1]- und W[2]-vollständiger Probleme Vertex-Cover, Clique und Hitting-Set untersucht. In allen Fällen lieferten diese Betrachtungen die Einsicht, dass Worst-Cases rar sind: Allein nach einem leichten Verrauschen werden die Probleme deutlich einfacher angreifbar als im Worst-Case. Dies zeigt die Stärke, die Techniken der parametrisierten Komplexität im Bereich der Average-Case- und Smoothed-Betrachtungen erbringen können.

Weiterführende Betrachtungen könnten ähnliche Ergebnisse bei anderen Problemen wie z. B. dem W[2]-vollständigen Dominating Set-Problem aufzeigen. Auch Härte-Aussagen und das Finden von Hierarchien wie durch [79] im parametrisierten Average-Case und ähnlich im Smoothed-Fall [13] angedeutet, könnten interessante nächste Schritte sein.

5 Probabilistische Greedy-Heuristiken

In diesem Kapitel soll anders als in den beiden vorhergehenden nicht die Eingabe einem probabilistischen Modell unterliegen, während der Algorithmus deterministisch formuliert werden kann. Hier betrachten wir stattdessen probabilistische Algorithmen, die auf feste Eingaben angewandt werden. Es stellt sich dabei heraus, dass diese Approximations-Algorithmen im Erwartungswert oft deutlich bessere Ergebnisse als ihre jeweiligen deterministischen Pendants erzeugen.

5.1 Allgemeines

Approximations-Algorithmen und die Frage, wie gut sich ein bestimmtes Problem überhaupt in polynomieller Zeit approximieren lässt, sind insbesondere für die notwendige Lösung in der Praxis auftauchender Probleme von großer Relevanz. Ein erster Ansatz ist dabei häufig der eines Greedy-Algorithmus, der jeweils die lokal beste Auswahl trifft. Für ganze Klassen verschiedener Probleme liefert ein solches Vorgehen auch immer eine optimale Lösung. Bei anderen hingegen, kann die Abweichung beliebig groß werden. Für die im weiteren Verlauf des Kapitels betrachteten Probleme geben wir in Tabelle 5.1 die entsprechenden Werte an.

In diesem Kapitel wollen wir die hier beschriebene klassische Herangehensweise der Verwendung von Greedy-Algorithmen insoweit verallgemeinern, als dass die Entscheidung, welches Element jeweils gerade ausgewählt und der zu konstruierenden Lösungsmenge hinzugefügt werden soll, nicht mehr deterministisch erfolgt, sondern einer noch zu konstruierenden Wahrscheinlichkeitsverteilung folgt. Da dieser Prozess dann zufällig ist, können verschiedene Lösungsmengen dadurch erhalten werden. Insofern ergibt es hier nur Sinn, über die erwartete Größe einer durch einen solchen probabilistischen Prozess erzeugten Lösungsmenge zu sprechen. Auch die so erwarteten Approximationsfaktoren (bzw. genauer: obere Abschätzungen dafür) finden sich für die im Folgenden betrachteten Probleme in Tabelle 5.1.

Es ist darauf hinzuweisen, dass die Idee, deterministische Greedy-Algorithmen durch ein Zufalls-Element effektiver zu gestalten, schon untersucht wurde. Dabei sind dem Autor allerdings nur Arbeiten bekannt, die entweder nicht nur das jeweils lokal beste Element, sondern alle, deren Güte oberhalb eines entsprechenden Schwellwerts liegen, berücksichtigen (und unter diesen eine gleichverteilte Auswahl treffen), siehe z. B. [14, 55, 91], oder das Problem als ganzzahliges Lineares Programm formulieren, eine nicht notwendigerweise ganzzahlige Lösung des relaxierten LPs bestimmen und dann die entsprechenden Variablen zufällig runden, z. B. [11, 85, 88].

Jedoch ist dem Autor keine Arbeit bekannt, die die im deterministischen Greedy-Algorithmus herangezogene und für die Auswahl der Elemente jeweils zu maximierende Größe (z. B. Knotengrad o. Ä.) direkt in eine Wahrscheinlichkeitsverteilung übersetzt, nach der dann das auszuwählende Element bestimmt wird. Dieser Ansatz wird deshalb in diesem Kapitel genauer an einigen Beispiel-Problemen untersucht.

	det. Greedy-Algo.		**prob. Greedy-Algo.**	
Vertex-Cover	$\Theta(\log n)$	[27]	2	Satz 5.2.2
d-Hitting-Set	$\Theta(\log n)$	[27]	d	Satz 5.3.2
Triangle-Vertex-Deletion	$\Theta(\log n)$	[27]	3	Satz 5.4.2
Dominating-Set mit maxdeg d	$\Theta(\log d)$	[27]	$d+1$	Satz 5.5.2

Tabelle 5.1: Überblick zur Approximationsgüte der deterministischen und in dieser Arbeit untersuchten probabilistischen Greedy-Approximations-Algorithmen für verschiedene Probleme. Bei den probabilistischen Algorithmen ist jeweils eine obere Abschätzung an den erwarteten Approximationsfaktor angegeben. Dieser gilt damit für alle Eingaben.

In den folgenden Abschnitten dieses Kapitels werden wir uns verschiedenen Minimierungsproblemen widmen. Bei diesen wird jeweils eine möglichst kleine Teilmenge einer Grundmenge gesucht, die eine bestimmte Eigenschaft besitzt. Da es sich dabei zumeist um Graph-Probleme handelt, sei deshalb in Algorithmus 5.1 exemplarisch nur von solchen die Rede.

Wir wollen nun diese (deterministischen) Greedy-Algorithmen verallgemeinern: Anstatt des jeweils lokal besten Elements soll in jedem Schritt ein Element gemäß einer (noch zu bestimmenden) Wahrscheinlichkeitsverteilung gezogen werden. In Algorithmus 5.1 wird ein solches Vorgehen schematisch dargestellt.

Algorithmus 5.1 : Allgemeiner Aufbau eines (probabilistischen) Greedy-Algorithmus für Graph-Minimierungs-Probleme

Input : Graph $G = (V, E)$ und eine Menge $Z \subset V$ zulässiger Knoten
Output : Eine Lösungsmenge $S \subset Z$

1 **if** *Erfüllt die leere Menge die problemspezifische Bedingung?* **then**
2 **return** $\emptyset$.
3 Bestimme eine Wahrscheinlichkeitsverteilung auf der Menge aller zulässiger Knoten Z.
4 Ziehe gemäß dieser Wahrscheinlichkeitsverteilung einen zulässigen Knoten $v \in Z$.
5 Reduziere problemspezifisch G und Z und löse dieses Subproblem rekursiv. Dessen Lösungsmenge sei S'.
6 **return** $S := S' \cup \{v\}$.

Beim Start-Aufruf von Algorithmus 5.1 wird üblicherweise die Menge Z der zulässigen Knoten identisch mit der gesamten Knotenmenge V des Graphen sein:

Jeder Knoten ist zulässig. Nur in die problemspezifischen Schritte 2 und 5 geht die Formulierung des zu lösenden Minimierungsproblems ein: Wann hat man eine Lösung gefunden bzw. welche Einschränkung an das Problem ergeben sich aus einer gefundenen Teillösung? Aber natürlich wird auch die Wahrscheinlichkeitsverteilung in Schritt 3 im Folgenden abhängig vom jeweils zu lösenden Problem gewählt werden.

Für die bekannten, deterministischen Greedy-Algorithmen ist diese Wahrscheinlichkeitsverteilung üblicherweise trivial: Ihre gesamte Konzentration liegt auf dem lokal besten, zulässigen Element. (Gibt es mehrere davon, so wird häufig eine Gleichverteilung, oder eine beliebige, deterministische Auswahl unter diesen angenommen.) Im Folgenden sollen nun andere Wahrscheinlichkeitsverteilungen betrachtet werden. Jedoch zeigt diese Überlegung, dass die probabilistischen Greedy-Algorithmen, wie sie in Algorithmus 5.1 beschrieben werden, eine Verallgemeinerung der wohluntersuchten deterministischen Greedy-Algorithmen darstellen.

In der weiteren Untersuchung werden wir dabei Wahrscheinlichkeitsverteilungen verwenden, sodass bei jedem Aufruf des Algorithmus mit mindestens einer festen Wahrscheinlichkeit p dabei ein Knoten, der zu einer optimalen Lösungsmenge gehört, gezogen wird. Dabei ist der Wert von p problemspezifisch und gegebenenfalls abhängig von weiteren Problem-Parametern. Da aber das Vorgehen in all diesen Fällen auf gleiche Weise zu den entsprechenden Resultaten über die erwartete Größe der gefundenen Lösungsmengen führt, wie wir sie in Tabelle 5.1 zusammengefasst haben, möchten wir dieses Vorgehen hier allgemein erläutern.

Dabei sei ab nun für eine Instanz, auf die Algorithmus 5.1 angewendet wird, eine optimale Lösungsmenge fixiert. Diese habe die Größe OPT. Wird in Schritt 4 ein Element dieser fixierten optimalen Lösungsmenge gezogen, so sprechen wir von einem „Erfolg“.

Lemma 5.1.1. *Wird in Algorithmus 5.1 eine Wahrscheinlichkeitsverteilung verwendet, die in jedem Aufruf sichert, dass mit mindestens der Wahrscheinlichkeit p ein Erfolg eintritt, so benötigt der Algorithmus im Erwartungswert höchstens $\frac{1}{p} \cdot OPT$ Rekursionsaufrufe, bis eine Lösungsmenge gefunden wurde. Der erwartete Approximationsfaktor lässt sich also nach oben durch $\frac{1}{p}$ abschätzen.*

Beweis. Zuerst berechnen wir die erwartete Anzahl von Rekursionsaufrufen bzw. „Versuchen“, bis ein Erfolg eingetreten ist. Da p eine untere Schranke für die Wahrscheinlichkeit in einem beliebigen Versuch für einen Erfolg ist, benötigt man mit Wahrscheinlichkeit von höchstens $p = \frac{p}{1-p} \cdot (1-p)^1$ genau einen, mit Wahrscheinlichkeit von höchstens $\frac{p}{1-p} \cdot (1-p)^2$ genau zwei, ... und mit Wahrscheinlichkeit von höchstens $\frac{p}{1-p} \cdot (1-p)^\ell$ genau ℓ Versuche, bis der erste Erfolg eintritt.

Der Erwartungswert der Anzahl der benötigten Versuche für den nächsten Erfolg beträgt also höchstens

$$\begin{aligned} E &\leq \sum_{\ell=1}^{\infty} \frac{p}{1-p} \cdot \ell \cdot (1-p)^{\ell} = \frac{p}{1-p} \cdot \sum_{t=0}^{\infty} \sum_{\ell=1}^{\infty} (1-p)^{\ell+t} \\ &= \frac{p}{1-p} \cdot \sum_{t=0}^{\infty} (1-p)^{t} \sum_{\ell=1}^{\infty} (1-p)^{\ell} = \frac{p}{1-p} \cdot \sum_{t=0}^{\infty} (1-p)^{t} \cdot \frac{1-p}{p} \\ &= \frac{p}{1-p} \cdot \frac{1-p}{p^2} = \frac{1}{p}. \end{aligned}$$

Nach erwarteten $\frac{1}{p}$ Versuchen hat man demnach einen zusätzlichen Erfolg; nach erwarteten $OPT \cdot \frac{1}{p}$ entsprechend OPT Erfolgsfälle. Damit ist in der so gefundenen die vorher fixierte optimale Lösungsmenge eine Teilmenge und deshalb selbst eine Lösung, die erwartet nur um den Faktor $\frac{1}{p}$ größer ist als das Optimum. □

Wichtig zu bemerken ist, dass diese Aussage über die erwartete Approximationsgüte des Algorithmus bezüglich der Instanzen eine Worst-Case-Abschätzung ist. Allein die Auswahl des jeweiligen Knotens in Schritt 4 von Algorithmus 5.1 enthält das Zufallselement in dieser Betrachtung.

Ist man mit einer $\frac{1}{p}$-Approximation noch nicht zufrieden, so kann man – unter Investition von entsprechend mehr Zeit – auch deutlich näher am Optimum liegende Lösungsmengen finden:

Lemma 5.1.2. *Sei $0 < p \leq 1$ und $1 \leq \alpha \leq \frac{1}{p}$. Wird in Algorithmus 5.1 eine Wahrscheinlichkeitsverteilung verwendet, die in jedem Aufruf sichert, dass mit mindestens der Wahrscheinlichkeit p ein Erfolg eintritt, so gibt es einen Wert OPT_{min}, der nur von p und α abhängt, sodass für alle Instanzen mit $OPT \geq OPT_{min}$ mit Wahrscheinlichkeit von mehr als $\frac{1}{2}$ unter den Ergebnissen von $3 \cdot \exp\left((-\log p) \cdot \frac{1-\alpha p}{1-p} \cdot OPT\right)$ Durchläufen des Algorithmus 5.1 sich eines befindet, dass nur um einen Faktor α größer ist als das Optimum OPT. Ist $p \leq \frac{1}{4}$, so genügen für die gleiche Wahrscheinlichkeitsaussage auch $4 \cdot \exp\left(2\frac{(1-\alpha p)^2}{\alpha p} \cdot OPT\right)$ Durchläufe.*

Wir bemerken, dass für alle Probleme, in denen das Optimum beschränkt ist, für die also das Lemma nur bedingte bis keine Aussage machen würde, ein einfacherer Algorithmus, der alle potentiellen Lösungsmengen durchprobiert, auch nur eine polynomielle Laufzeit hätte. Die entsprechenden Probleme liegen also in P und müssen hier nicht weiter betrachtet werden, da sie in polynomieller Zeit exakt gelöst werden können und nicht approximiert werden müssen. Daher können wir für die weitere Betrachtung annehmen, dass die Instanzen der hier betrachteten Probleme Folgen von Instanzen besitzen, deren optimale Lösungsmengen unbeschränkt wachsen. Damit ist das Lemma 5.1.2 von entsprechender Bedeutung.

Weiterhin ist wegen

$$\begin{aligned}
(-\log p)\cdot\frac{1-\alpha p}{1-p} &\geq 2\frac{(1-\alpha p)^2}{\alpha p}\\
\Leftrightarrow(-\log p)\cdot\frac{1}{1-p} &\geq 2\cdot\frac{1-\alpha p}{\alpha p}\\
\Leftrightarrow(-\log p)\cdot\frac{\alpha p}{1-p} &\geq 2\cdot(1-\alpha p)\\
\Leftrightarrow(-\log p)\cdot\frac{p}{1-p} &\geq 2\cdot\left(\frac{1}{\alpha}-p\right)
\end{aligned}$$

für festes p und genügend große $\alpha < \frac{1}{p}$ die zweite Angabe in Lemma 5.1.2 der benötigten Durchläufe eine echte Verschärfung der ersten.

Beweis. Zuerst bestimmen wir die Wahrscheinlichkeit P, dass ein beliebiger Durchlauf von Algorithmus 5.1 zu einer α-Approximation führt:

Dazu betrachten wir die vom Algorithmus $x = \frac{\alpha-1}{1-p}\cdot OPT$ zuerst ausgewählten Elemente. Wir bemerken, dass wegen $\alpha \leq \frac{1}{p}$ und damit $p \leq \frac{1}{\alpha}$ bzw. $1-p \geq 1-\frac{1}{\alpha} = \frac{\alpha-1}{\alpha}$ dieser Wert $x \leq \alpha\cdot OPT$ ist. Dass sich bei der Auswahl dieser x Elemente darunter mindestens $p\cdot x$ Erfolge befinden, geschieht mit einer Wahrscheinlichkeit, die für unbeschränkt wachsendes OPT (und damit auch, bei fest gewählten Parametern p und α, unbeschränkt wachsendes x) gegen $\frac{1}{2}$ geht: Die zugrundeliegende Binomialverteilung, die die Zahl der Erfolge misst, nähert sich immer weiter der Normalverteilung an. Und für diese ist aus Symmetriegründen die Wahrscheinlichkeit, mindestens die erwartete Anzahl $p\cdot x$ an Erfolgen zu erhalten, $\frac{1}{2}$. Es gibt demnach einen Wert OPT_{min}, der nur von p und α abhängt, sodass für alle $OPT \geq OPT_{min}$ die Wahrscheinlichkeit, dass unter x Versuchen mindestens $p\cdot x$ Erfolge sind, mindestens $\frac{1}{3}$ beträgt.

Um eine Lösungsmenge zu erhalten, fehlen also dann nur noch höchstens

$$\begin{aligned}
y &= OPT - p\cdot x = OPT\cdot\left(1-p\cdot\frac{\alpha-1}{1-p}\right) = OPT\cdot\frac{(1-p)-p\cdot(\alpha-1)}{1-p}\\
&= OPT\cdot\frac{1-\alpha\cdot p}{1-p}
\end{aligned}$$

weitere Erfolge. Dass nun jedes der folgenden y von Algorithmus 5.1 ausgewählten Elemente einen Erfolg liefert, geschieht offenbar mindestens mit der Wahrscheinlichkeit p^y. Insgesamt erhalten wir also eine Lösungsmenge, die mit Wahrscheinlichkeit $P \geq \frac{1}{3}\cdot p^{\frac{1-\alpha p}{1-p}OPT}$ aus höchstens $x+y = \left(\frac{\alpha-1}{1-p}+\frac{1-\alpha\cdot p}{1-p}\right)\cdot OPT = \alpha\cdot OPT$ Elementen besteht.

Wird der Algorithmus 5.1 also $\frac{1}{P} = 3 \cdot \left(\frac{1}{p}\right)^{\frac{1-\alpha p}{1-p} \cdot OPT} = 3 \cdot \exp\left((-\log p) \cdot \frac{1-\alpha p}{1-p} \cdot OPT\right)$ mal wiederholt ausgeführt, so ist die Wahrscheinlichkeit, dass in keinem dieser Durchläufe eine α-Approximation gefunden wird, höchstens $1 - (1-P)^{\frac{1}{P}}$, was für alle $P > 0$ selbst größer ist als $1 - \frac{1}{e} > \frac{1}{2}$. Damit ist der erste Teil der Aussage bewiesen.

Ist dagegen $p \leq \frac{1}{4}$, so können wir mit der in Lemma 2.8.2 angegebenen inversen Chernoff-Schranke ein noch leicht besseres Ergebnis erhalten. Die Wahrscheinlichkeit P, bei einem Durchlauf von Algorithmus 5.1 eine α-Approximation zu erhalten, lässt sich dann wie folgt nach unten abschätzen:

Bei $\alpha \cdot OPT$ von Algorithmus 5.1 ausgewählten Elementen liegt die erwartete Anzahl an Erfolgen bei mindestens $\mu = (\alpha p) \cdot OPT$. Da aber OPT Erfolge notwendig sind, damit wir garantieren können, dass eine Lösungsmenge gefunden wurde, muss die Anzahl der Erfolge um den Faktor $1 + \delta = \frac{1}{\alpha p}$ größer sein. Setzen wir dies nun in Lemma 2.8.2 ein, so erhalten wir mit $\delta^2 \mu = \frac{(1-\alpha p)^2}{\alpha p} \cdot OPT$, dass schon mit der Wahrscheinlichkeit $P \geq \frac{1}{4} \exp\left(-2\frac{(1-\alpha p)^2}{\alpha p} \cdot OPT\right)$ bei einem Durchlauf eine α-Approximation gefunden wird. Also genügen mit dem gleichen Gedanken wie oben $4 \cdot \exp\left(2\frac{(1-\alpha p)^2}{\alpha p} \cdot OPT\right)$ Durchläufe, um mit Wahrscheinlichkeit $\frac{1}{2}$ eine α-Approximation zu finden. □

Abschließend bemerken wir, dass man durch das Setzen von $\alpha = 1$ im Lemma 5.1.2 damit auch einen einfachen randomisierten FPT-Algorithmus für das betrachtete Problem erhält:

Lemma 5.1.3. *Sei $k \in \mathbb{N}$ und $0 < p \leq 1$. Wird in Algorithmus 5.1 eine Wahrscheinlichkeitsverteilung verwendet, die in jedem Aufruf sichert, dass mit mindestens der Wahrscheinlichkeit p ein Erfolg eintritt, so wird für alle Instanzen, in denen eine Lösungsmenge der Größe höchstens k existiert, diese in einer erwarteten Anzahl von höchstens $\left(\frac{1}{p}\right)^k$ Durchläufen von Algorithmus 5.1 gefunden. Da ein solcher nur polynomielle Laufzeit in Abhängigkeit der Eingabegröße besitzt, erhält man damit insgesamt eine erwartete* FPT*-Laufzeit.*

Beweis. Die Wahrscheinlichkeit P, eine Lösung der Größe höchstens k in einem Durchlauf von Algorithmus 5.1 zu finden, ist durch $P \geq p^k$ gegeben: Bei jedem vom Algorithmus ausgewählten Element besteht nach Voraussetzung die Wahrscheinlichkeit von mindestens p Teil der gesuchten Lösungsmenge zu sein. Da diese Wahrscheinlichkeiten unabhängig voneinander sind, multiplizieren sie sich zur Abschätzung von P.

Damit ist die erwartete Anzahl von Durchläufen, bis einer die gesuchte Lösungsmenge findet, höchstens $\frac{1}{P}$, was das Lemma beweist. □

5.2 Das Vertex-Cover-Problem

Beim Vertex-Cover-Problem, welches schon zuvor in dieser Arbeit betrachtet wurde, soll eine möglichst kleine Teilmenge der Knoten eines Graphen ausgewählt werden, sodass jede Kante im Graphen mindestens einen Endpunkt in der ausgewählten Knoten-Teilmenge besitzt.

Da also mit möglichst wenigen Knoten möglichst viele Kanten abgedeckt werden sollen, wählt der klassische (deterministische) Greedy-Algorithmus in jedem Schritt einen Knoten, der maximal viele neue Kanten abdeckt, aus. Dies verallgemeinern wir dahingehend, dass nun ein Knoten gemäß einer Wahrscheinlichkeitsverteilung ausgewählt wird, die proportional zur Anzahl der vom jeweiligen Knoten neu abgedeckten Kanten ist. Algorithmus 5.2 als Spezialisierung von Algorithmus 5.1 gibt dies noch einmal in detaillierter Form wieder.

Algorithmus 5.2 : Probabilistischer Greedy-Algorithmus für das Vertex-Cover-Problem

Input : Graph $G = (V, E)$
Output : Eine Knotenüberdeckung $S \subset V$

1 **if** *Sind im Graphen keine weiteren Kanten mehr zu überdecken?* **then**
2 **return** $\emptyset$.
3 Bestimme für jeden Knoten die Anzahl der noch nicht überdeckten Kanten, deren Endpunkt er ist.
4 Ziehe gemäß der Wahrscheinlichkeitsverteilung, die proportional zu den eben bestimmten Anzahlen ist, einen zulässigen Knoten $v \in V$.
5 Lösche den Knoten v sowie alle seine Kanten aus G und löse dieses Subproblem rekursiv. Dessen Lösungsmenge sei S'.
6 **return** $S := S' \cup \{v\}$.

Zu bemerken ist dabei, dass in Algorithmus 5.2 immer alle noch vorhandenen Knoten zulässig sind. Deshalb wurde konsequent für die Menge Z der zulässigen Knoten in Algorithmus 5.1 die gesamte Knotenmenge V eingesetzt. Auch geben wir in Schritt 3 nicht direkt die Wahrscheinlichkeitsverteilung an, sondern ermitteln für jeden zulässigen Knoten ein nicht-negatives Gewicht (hier der Knotengrad) und legen die Wahrscheinlichkeitsverteilung dann so fest, dass die Auswahlwahrscheinlichkeit eines Knotens proportional zu seinem Gewicht ist. Dies werden wir aufgrund der einfacheren Formulierung auch in den folgenden Abschnitten beibehalten.

Damit wir das Lemma 5.1.1 anwenden können, müssen wir zeigen, dass diese Wahrscheinlichkeitsverteilung die Eigenschaft erfüllt, dass in jedem Schritt die Wahrscheinlichkeit, einen Erfolg zu erzielen (d. h., einen Knoten der fixierten optimalen Lösungsmenge zu ziehen), durch einen Wert $p > 0$ abgeschätzt werden kann. Dies soll im folgenden Lemma geschehen.

Lemma 5.2.1. *Sei G ein Graph und M eine fixierte, optimale Knotenüberdeckung von G. Dann ist für jeden Rekursions-Aufruf von Algorithmus 5.2, der nicht die leere Menge als Lösung zurückgibt, die Wahrscheinlichkeit, ein Element aus M zu ziehen, mindestens $\frac{1}{2}$.*

Beweis. Es sei g die Summe der Gewichte der bisher noch nicht gezogenen Elemente der fixierten, optimalen Knotenüberdeckung M und s die Summe der Gewichte aller (zulässigen) Knoten. Dann beträgt die Wahrscheinlichkeit dafür, eines der Elemente aus M zu ziehen und mithin einen Erfolg zu erzielen, also $p = \frac{g}{s}$.

Da die noch fehlenden Elemente aus M weiterhin alle Kanten überdecken, muss also $g \geq |E|$ gelten. Umgekehrt wird aber jede Kante von genau zwei Knoten überdeckt, sodass die Summe s der Knotengrade über alle Knoten des Graphen jede Kante genau zweimal zählt. Also ist $s = 2 \cdot |E|$.

Setzen wir dies ein, erhalten wir $p \geq \frac{|E|}{2 \cdot |E|} = \frac{1}{2}$. □

Damit folgt der erste Satz über die Approximationsgüte eines probabilistischen Greedy-Algorithmus:

Satz 5.2.2. *Der Algorithmus 5.2 besitzt einen erwarteten Approximationsfaktor von $\alpha = 2$. Das heißt, die von Algorithmus 5.2 gefundene Knotenüberdeckung ist im Erwartungswert höchstens doppelt so groß wie eine optimale.*

Bemerkenswert ist dies, da der deterministische Greedy-Algorithmus nur einen garantierten Approximationsfaktor von $\Theta(\log |V|)$ besitzt, siehe zum Beispiel [27] und [64]. Numerische Simulationen lassen vermuten, dass für Graphen mit $n \geq 2$ Knoten der im Worst-Case eintretende Approximationsfaktor von Algorithmus 5.2 den Wert $2 - 2^{-(n-2)}$ besitzt.

Beweis. Nach Lemma 5.2.1 ist in jedem nicht-trivialen Durchlauf von Algorithmus 5.2 die Wahrscheinlichkeit, einen Erfolg zu erzielen, mindestens $p = \frac{1}{2}$. Wenden wir in dieser Situation nun das Lemma 5.1.1 an, erhalten wir den erwarteten Approximationsfaktor von $\frac{1}{p} = 2$, was die zu beweisende Aussage zeigt. □

Aber man kann ja auch – mit etwas mehr nötiger Laufzeit – bessere Approximationen erzeugen:

Satz 5.2.3. *Sei $1 \leq \alpha < 2$. Dann findet Algorithmus 5.2 mit Wahrscheinlichkeit von mehr als $\frac{1}{2}$ in höchstens $3 \cdot \exp\left((2 \log 2) \cdot \left(1 - \frac{\alpha}{2}\right) \cdot OPT\right)$ Durchläufen eine Lösungsmenge der Größe von höchstens $\alpha \cdot OPT$.*

Beweis. Wendet man Lemma 5.1.2 mit $p = \frac{1}{2}$ an, folgt sofort das gewünschte Ergebnis. □

Dabei ist zu bemerken, dass das Vertex-Cover-Problem mit deterministischen Algorithmen nicht besser als mit dem Faktor $1,36$ approximiert werden kann, es sei denn $\mathsf{P} = \mathsf{NP}$, [33]. Mit Satz 5.2.3 jedoch erhalten wir – aufgrund der polynomiellen Laufzeit eines Durchlaufs von Algorithmus 5.2 – einen randomisierten FPT-Approximations-Algorithmus, der beliebig genaue Näherungen einer optimalen Lösung liefert. Dass dergleichen überhaupt existiert, mag bei der Existenz von (Worst-Case)-FPT-Algorithmen für dieses Problem, die sogar das Optimum finden, nicht allzu sehr verwundern. Da die erwartete Laufzeit der hier angegebenen Vorgehensweise für $\alpha \to 2$ auf eine polynomielle (und sogar lineare) Funktion fällt, stellt sie, wenn auch etwas gröbere Näherungen ausreichen – zum Beispiel für eine Initialisierung weiterer Prozesse – eine (im Erwartungswert) schnellere Alternative als die bekannten (deterministischen) FPT-Algorithmen dar.

Wir weisen darauf hin, dass eine ähnliche Aussage sich in [49] für das hier betrachtete Vertex-Cover-Problem findet.

Das Vorgehen in Algorithmus 5.2 ist dabei äquivalent zur gleichverteilten Auswahl einer bisher noch unüberdeckten Kante und dann wiederum gleichverteilt einer ihrer beiden Endknoten. Während dieser Ansatz im Sinne eines beschränkten Suchbaums nur eine parametrisierte Laufzeit von $\mathcal{O}(2^k \cdot n)$ erhalten lässt, wobei $k = OPT$ die Größe einer optimalen Knotenüberdeckung ist, liefert ein elaborierterer Ansatz, je nach Ausgestaltung parametrisierte Laufzeiten, die deutlich kleiner sind, z. B. in $\mathcal{O}(1,2738^k \cdot n)$, [26]. Dabei wird insbesondere danach verzweigt, ob ein Knoten oder alle seine Nachbarn Teil der Lösungsmenge sind. Konstruiert man aber damit ähnlich dem hier gezeigten einen probabilistischen Greedy-Algorithmus, so liefert dieser interessanter Weise auch nur einen erwarteten Approximationsfaktor von 2. Da die entsprechende Analyse also keine neuen, besseren Ergebnisse liefert, soll hier nicht weiter darauf eingegangen werden.

5.3 Das d-Hitting-Set-Problem

Auch das d-Hitting-Set-Problem wurde im vorherigen Kapitel schon betrachtet. Hierbei ist eine Grundmenge G und eine Mengenfamilie $\mathcal{F}$ von Teilmengen von G gegeben, wobei jede dieser Teilmengen jeweils d Elemente aus der Grundmenge enthält. Gesucht ist nun eine möglichst kleine Teilmenge S von G, sodass jedes Element aus $\mathcal{F}$ mit S einen nichtleeren Schnitt besitzt.

Ein deterministischer Greedy-Algorithmus würde hierbei in jedem Durchlauf das Element der Grundmenge auswählen, welches in den meisten bisher noch nicht getroffenen Elementen von $\mathcal{F}$ vorkommt. Auch dies wollen wir wieder im Sinne einer stochastischen Auswahl verallgemeinern, indem die Wahrscheinlichkeitsverteilung proportional zu eben jener Zahl an neu durch das Element getroffenen Mengen der Mengenfamilie $\mathcal{F}$ gewählt wird.

Damit verfolgen wir, obwohl es sich in dieser Formulierung nicht um ein Graph-Problem handelt, einen Ansatz wie im allgemeinen Algorithmus 5.1 bzw. sogar analog zum Algorithmus 5.2, der das VERTEX-COVER-Problem löst. Dies kommt auch nicht von ungefähr, denn für $d = 2$ kann das HITTING-SET-Problem auch als die Suche nach einer Minimum-Knotenüberdeckung gedeutet werden. Algorithmus 5.3 gibt die Details an.

Algorithmus 5.3 : Probabilistischer Greedy-Algorithmus für das HITTING-SET-Problem

Input : Grundmenge G und Mengenfamilie $\mathcal{F} \subset \mathcal{P}(G)$
Output : Eine Teilmenge $S \subset G$, sodass jedes Element $X \in \mathcal{F}$ nichtleeren Schnitt mit S besitzt.

1 **if** *Ist* $\mathcal{F} = \emptyset$*?* **then**
2 **return** $\emptyset$.
3 Bestimme für jedes Element der Grundmenge die Anzahl der Mengen aus $\mathcal{F}$, deren Element es ist.
4 Ziehe gemäß der Wahrscheinlichkeitsverteilung, die proportional zu den eben bestimmten Anzahlen ist, ein Element $v \in G$.
5 Lösche das Element v aus G sowie alle Mengen, die es als Element enthalten, aus $\mathcal{F}$ und löse dieses Subproblem rekursiv. Dessen Lösungsmenge sei S'.
6 **return** $S := S' \cup \{v\}$.

Der Algorithmus 5.3 löst dabei das generelle HITTING-SET-Problem. Die Einschränkung, dass die Mengen in $\mathcal{F}$ jeweils genau d Elemente besitzen sollen, ist nur für die Bestimmung der Approximationsgüte notwendig. Diese wollen wir im Folgenden ermitteln und folgen dem Vorgehen aus Abschnitt 5.2.

Lemma 5.3.1. *Sei G eine Menge, $\mathcal{F}$ eine Familie von Teilmengen mit je d verschiedenen Elementen von G und M eine fixierte, optimale Teilmenge von G, sodass jedes Element aus $\mathcal{F}$ mit M einen nichtleeren Schnitt besitzt. Dann ist für jeden Rekursions-Aufruf von Algorithmus 5.3, der nicht die leere Menge als Lösung zurückgibt, die Wahrscheinlichkeit, ein Element aus M zu ziehen, mindestens $\frac{1}{d}$.*

Beweis. Wie in Lemma 5.2.1 sei g die Summe der Gewichte der bisher noch nicht gezogenen Elemente der fixierten, optimalen Lösungsmenge M und s die Summe der Gewichte aller (zulässigen) Elemente der Grundmenge. Dann beträgt die Wahrscheinlichkeit dafür, eines der Elemente aus M zu ziehen und mithin einen Erfolg zu erzielen, also $p = \frac{g}{s}$.

Da die noch fehlenden Elemente aus M weiterhin alle Mengen aus der Familie $\mathcal{F}$ treffen, muss also $g \geq |\mathcal{F}|$ gelten. Umgekehrt wird aber jede dieser Mengen von genau d Elementen der Grundmenge getroffen, sodass die Summe s der Gewichte aller Elemente jede dieser Mengen in $\mathcal{F}$ genau d-mal zählt. Demnach ist $s = d \cdot |\mathcal{F}|$.

Setzen wir dies ein, erhalten wir $p \geq \frac{|\mathcal{F}|}{d \cdot |\mathcal{F}|} = \frac{1}{d}$. □

Satz 5.3.2. *Der Algorithmus 5.3 besitzt bei Aufrufen, bei der alle Mengen der Familie $\mathcal{F}$ jeweils genau d Elemente besitzen, einen erwarteten Approximationsfaktor von $\alpha = d$. Das heißt, die von Algorithmus 5.3 gefundene Lösungsmenge ist im Erwartungswert höchstens d mal so groß wie eine optimale.*

Beweis. Nach Lemma 5.3.1 ist in jedem nicht-trivialen Durchlauf von Algorithmus 5.3 die Wahrscheinlichkeit, einen Erfolg zu erzielen, mindestens $p = \frac{1}{d}$. Wenden wir in dieser Situation nun das Lemma 5.1.1 an, erhalten wir den erwarteten Approximationsfaktor von $\frac{1}{p} = d$. □

Und wieder können wir – unter Benutzung von mehr Laufzeit – bessere Approximationen erzeugen:

Satz 5.3.3. *Sei $1 \leq \alpha < d$. Dann findet Algorithmus 5.3 mit Wahrscheinlichkeit von mehr als $\frac{1}{2}$ in höchstens $3 \cdot \exp\left((d \log d) \cdot \left(1 - \frac{\alpha}{d}\right) \cdot OPT\right)$ Durchläufen eine Lösungsmenge der Größe von höchstens $\alpha \cdot OPT$.*

Beweis. Wendet man Lemma 5.1.2 mit $p = \frac{1}{d}$ an, folgt sofort das gewünschte Ergebnis. □

Die Voraussetzung der Sätze 5.3.2 und 5.3.3 bzw. Lemma 5.3.1, dass die Mengen der Familie jeweils genau d Elemente besitzen müssen, kann auch ohne Einschränkung zu „höchstens d Elemente“ abgeschwächt werden. Damit ergibt sich also eine entsprechende Aussage für das allgemeine Hitting-Set-Problem mit dem Parameter „Mächtigkeit der größten Menge“.

5.4 Das Triangle-Vertex-Deletion-Problem

Beim Triangle-Vertex-Deletion-Problem ist ein Graph gegeben und es wird eine möglichst kleine Teilmenge seiner Knoten gesucht, nach deren Löschen (mitsamt ihrer Kanten) der nun kleinere Graph kein Dreieck, d. h. K_3, mehr als Teilgraphen enthält. Dies kann man als eine Variation des Vertex-Cover-Problems ansehen, da dort nach dem Löschen der Lösungsmenge keine Kante, d. h. also K_2, mehr vorhanden sein soll.

In einem deterministischen Greedy-Algorithmus bestimmt man in jedem Durchlauf denjenigen Knoten, dessen Löschen die meisten Dreiecke eliminiert. Auch dies wird jetzt wieder verallgemeinert, indem die Anzahl der Dreiecke, an denen ein Knoten beteiligt ist, als Gewicht gewählt wird, zu dem dann seine Auswahlwahrscheinlichkeit proportional sein soll. Algorithmus 5.4 enthält die Details.

Auch die in Algorithmus 5.4 verwendete Wahrscheinlichkeitsverteilung soll analog dem Vorgehen der vorherigen Abschnitte analysiert werden:

Algorithmus 5.4 : Probabilistischer Greedy-Algorithmus für das Triangle-Vertex-Deletion-Problem

Input : Graph $G = (V, E)$
Output : Eine Teilmenge $S \subset V$ der Knoten, nach deren Löschung G keinen K_3 mehr enthält.

1 **if** *Sind im Graphen keine weiteren Dreiecke mehr enthalten?* **then**
2 **return** $\emptyset$.
3 Bestimme für jeden Knoten die Anzahl der Dreiecke, an denen er beteiligt ist.
4 Ziehe gemäß der Wahrscheinlichkeitsverteilung, die proportional zu den eben bestimmten Anzahlen ist, einen zulässigen Knoten $v \in V$.
5 Lösche den Knoten v sowie alle seine Kanten aus G und löse dieses Subproblem rekursiv. Dessen Lösungsmenge sei S'.
6 **return** $S := S' \cup \{v\}$.

Lemma 5.4.1. *Sei $G = (V, E)$ ein Graph und M eine fixierte, optimale Teilmenge von V, sodass der von $V \setminus M$ erzeugte Teilgraph von G keinen K_3 mehr enthält. Dann ist für jeden Rekursions-Aufruf von Algorithmus 5.4, der nicht die leere Menge als Lösung zurückgibt, die Wahrscheinlichkeit, ein Element aus M zu ziehen, mindestens $\frac{1}{3}$.*

Beweis. Wie in Lemma 5.2.1 sei g die Summe der Gewichte der bisher noch nicht gezogenen Elemente der fixierten, optimalen Lösungsmenge M und s die Summe der Gewichte aller Knoten des Graphen. Dann beträgt die Wahrscheinlichkeit dafür, eines der Elemente aus M zu ziehen und mithin einen Erfolg zu erzielen, wieder $p = \frac{g}{s}$.

Da die noch fehlenden Elemente aus M weiterhin alle Dreiecke abdecken, muss also $g \geq |\mathcal{D}|$ gelten, wobei $\mathcal{D}$ die Menge der in G enthaltenen Dreiecke sei. Umgekehrt wird aber jede dieser Dreiecke von genau drei Knoten gebildet, sodass die Summe s der Gewichte aller Knoten jedes Dreieck in $\mathcal{D}$ genau dreimal zählt. Also ist $s = 3 \cdot |\mathcal{D}|$.

Setzen wir dies ein, erhalten wir $p \geq \frac{|\mathcal{D}|}{3 \cdot |\mathcal{D}|} = \frac{1}{3}$. □

Satz 5.4.2. *Der Algorithmus 5.4 besitzt einen erwarteten Approximationsfaktor von $\alpha = 3$. Das heißt, die von Algorithmus 5.4 gefundene Lösungsmenge ist im Erwartungswert höchstens dreimal so groß wie eine optimale.*

Beweis. Nach Lemma 5.4.1 ist in jedem nicht-trivialen Durchlauf von Algorithmus 5.4 die Wahrscheinlichkeit, einen Erfolg zu erzielen, mindestens $p = \frac{1}{3}$. Wenden wir in dieser Situation nun das Lemma 5.1.1 an, erhalten wir den erwarteten Approximationsfaktor von $\frac{1}{p} = 3$. □

Auch hier können wir wieder den Tradeoff zwischen besserer Approximationsgüte und erwarteter Laufzeit erzeugen:

Satz 5.4.3. *Sei $1 \leq \alpha < 3$. Dann findet Algorithmus 5.3 mit Wahrscheinlichkeit von mehr als $\frac{1}{2}$ in höchstens $3 \cdot \exp\left((3 \log 3) \cdot \left(1 - \frac{\alpha}{3}\right) \cdot OPT\right)$ Durchläufen eine Lösungsmenge der Größe von höchstens $\alpha \cdot OPT$.*

Beweis. Wendet man Lemma 5.1.2 mit $p = \frac{1}{3}$ an, folgt sofort das gewünschte Ergebnis. □

Während das Triangle-Vertex-Deletion-Problem sich noch direkt auf das 3-Hitting-Set-Problem reduzieren lässt (die Grundmenge ist dabei die Knotenmenge und die Mengenfamilie die Menge aller Dreiecke im Graphen), ist dies für das allgemeine Subgraph-Vertex-Deletion-Problem nicht mehr der Fall. Jedoch lässt sich auch hier der probabilistische Greedy-Algorithmus entsprechend anwenden und man erhält auf analoge Weise im Erwartungswert eine a-Approximation, wenn der zu entfernende Subgraph a Knoten besitzt.

5.5 Das Dominating-Set-Problem in Graphen mit Maximalgrad d

Der folgende Abschnitt widmet sich dem Dominating-Set-Problem. Dabei ist ein Graph $G = (V, E)$ gegeben und es ist eine möglichst kleine Teilmenge $S \subset V$ der Knotenmenge gesucht, sodass jeder weitere Knoten in $V \setminus S$ mit mindestens einem der Knoten aus S durch eine Kante verbunden ist. Man sagt auch, dass ein Knoten dabei seine abgeschlossene Nachbarschaft (d. h. inklusive sich selbst) dominiert.

Ein deterministischer Greedy-Algorithmus wählt nun in jedem Schritt den Knoten aus, der die meisten neuen Knoten dominiert. Wieder verallgemeinern wir diesen Ansatz, indem wir den Knoten mit Wahrscheinlichkeit proportional zu dieser Anzahl auswählen. Algorithmus 5.5 geht dabei genauer ins Detail.

Initial wird Algorithmus 5.5 mit der Menge der noch nicht dominierten Knoten $D = V$ aufgerufen. Er löst allgemein das Dominating-Set-Problem. Um jedoch geeignete Aussagen über die in Schritt 4 verwendete Wahrscheinlichkeitsverteilung gemäß der in Schritt 3 ermittelten Gewichte zu erhalten, werden wir im Folgenden nur eine Einschränkung an die Eingabe-Graphen betrachten: Graphen mit einem vorgegebenen Maximalgrad d.

Lemma 5.5.1. *Sei $G = (V, E)$ ein Graph mit Maximalgrad d und M eine fixierte, optimale dominierende Teilmenge von V, d. h., jeder Knoten in $V \setminus M$ besitzt mindestens einen Nachbarn in M und M hat dabei die dafür minimal mögliche Größe. Dann ist für jeden Rekursions-Aufruf von Algorithmus 5.5, der nicht die leere Menge als Lösung zurückgibt, die Wahrscheinlichkeit, ein Element aus M zu ziehen, mindestens $\frac{1}{d+1}$.*

Algorithmus 5.5 : Probabilistischer Greedy-Algorithmus für DOMINATING-SET

Input : Graph $G = (V, E)$ und eine Menge $D \subset V$ der noch nicht dominierten Knoten

Output : Eine Teilmenge $S \subset V$ der Knoten, die alle Knoten aus D dominiert.

1 **if** *Ist* $D = \emptyset$*?* **then**
2 **return** $\emptyset$.
3 Bestimme für jeden Knoten aus V die Anzahl der Knoten aus D, die dieser dominiert.
4 Ziehe gemäß der Wahrscheinlichkeitsverteilung, die proportional zu den eben bestimmten Anzahlen ist, einen zulässigen Knoten $v \in V$.
5 Lösche den Knoten v aus G sowie alle durch ihn dominierten (inkl. ihm selbst) aus D und löse dieses Subproblem rekursiv. Dessen Lösungsmenge sei S'.
6 **return** $S := S' \cup \{v\}$.

Beweis. Wie in Lemma 5.2.1 sei g die Summe der Gewichte der bisher noch nicht gezogenen Elemente der fixierten, optimalen Lösungsmenge M und s die Summe der Gewichte aller Knoten des Graphen. Dann beträgt die Wahrscheinlichkeit dafür, eines der Elemente aus M zu ziehen und mithin einen Erfolg zu erzielen, wieder $p = \frac{g}{s}$.

Die Menge der noch zu dominierenden Knoten sei mit D bezeichnet. Da die noch fehlenden Elemente aus M weiterhin alle Knoten aus D abdecken, muss also $g \geq |D|$ gelten. Umgekehrt kann jeder der Knoten aus D nur von sich und seinen höchstens d Nachbarn dominiert werden, sodass die Summe s aller Gewichte durch $s \leq (d+1) \cdot |D|$ beschränkt ist.

Setzen wir dies ein, erhalten wir $p \geq \frac{|D|}{(d+1) \cdot |D|} = \frac{1}{d+1}$. □

Satz 5.5.2. *Der Algorithmus 5.5 besitzt einen erwarteten Approximationsfaktor von* $\alpha = (d + 1)$*. Das heißt, die von Algorithmus 5.5 gefundene Lösungsmenge ist im Erwartungswert höchstens* $d + 1$*-mal so groß wie eine optimale.*

Beweis. Nach Lemma 5.5.1 ist in jedem nicht-trivialen Durchlauf von Algorithmus 5.4 die Wahrscheinlichkeit, einen Erfolg zu erzielen, mindestens $p = \frac{1}{(d+1)}$. Wenden wir in dieser Situation nun das Lemma 5.1.1 an, erhalten wir den erwarteten Approximationsfaktor von $\frac{1}{p} = (d + 1)$. □

Damit konnten wir hier kein besseres Ergebnis, als für den deterministischen Greedy-Algorithmus bekannt ist, zeigen. Denn dieser liefert bei Eingabe-Graphen mit Maximalgrad d in jedem Fall Approximationslösungen, die nur um einen Faktor von höchstens $\Theta(\log d)$ größer sind als das Optimum.

Jedoch konnten numerische Simulationen die Vermutung untermauern, dass die in Algorithmus 5.5 angegebene probabilistische Variante des deterministischen Greedy-Algorithmus im Erwartungswert nur um einen konstanten Faktor schlechtere Lösungen als der deterministische erzeugt.

Dazu wurden auf einer Vielzahl von Erdős-Rényi-Graphen mit verschiedenen Kantenwahrscheinlichkeiten von $p = 0,02$ bis $0,5$ und Knotenanzahlen von 10 bis 1000 jeweils mittels des deterministischen und dem hier angegebenen probabilistischen Greedy-Algorithmus eine dominierende Menge in diesen Graphen bestimmt. Um für den probabilistischen Algorithmus den Erwartungswert der Größe der gefundenen dominierenden Menge näherungsweise zu erhalten, wurden auf dem gleichen Graphen jeweils 100 Durchläufe von Algorithmus 5.5 durchgeführt und deren Ergebnisse gemittelt. Es zeigt sich, dass dabei nie die erhaltene „durchschnittliche“ Lösung von Algorithmus 5.5 um einen Faktor 2 oder mehr größer war als die des deterministischen Greedy-Algorithmus. Dies ist deutlich unter dem Faktor $\Theta\left(\frac{d}{\log d}\right)$, mit dem Maximalgrad d des betrachteten Graphen, welcher sich ergeben würde, wenn die Aussage von Satz 5.5.2 scharf wäre. Offenbar gilt wohl eine viel stärkere Aussage.

Dies würde aber einerseits bedeuten, dass man mit genügend vielen Wiederholungen in FPT-Laufzeit bessere Approximationsergebnisse als mit dem deterministischen Greedy-Algorithmus erzielen könnte; und sogar mit Lemma 5.1.3 eine Möglichkeit für einen randomisierten FPT-Algorithmus für das Dominating-Set-Problem erhalten würde. Dies ist in zukünftigen Überlegungen genauer zu untersuchen.

5.6 Zusammenfassung

Wir haben gezeigt, dass eine einfache Verallgemeinerung der simplen Greedy-Strategie auf einen probabilistischen Ansatz bei vielen Problemen im Erwartungswert deutlich bessere Näherungslösungen als die jeweilige deterministische Variante erzeugt. Zwar existieren auch andere, bessere deterministische Approximationsalgorithmen als allein die Verwendung der Greedy-Eigenschaft. Doch auch diese haben gegenüber den hier vorgestellten probabilistischen Algorithmen den Nachteil, dass sich ihre Ergebnisse durch Wiederholung nicht verbessern lassen.

Damit haben wir hier auch eine einfache Strategie zur Konstruktion von effizienten randomisierten FPT-Approximations-Algorithmen aufgezeigt. Weitere Probleme und auch die Verallgemeinerung anderer deterministischer Approximations-Algorithmen auf probabilistische Weise bieten sich als zukünftige Untersuchungsgegenstände an.

Literaturverzeichnis

[1] P. K. Agarwal, N. Alon, B. Aronov und S. Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12:347–365, 1994.

[2] M. Agrawal, N. Kayal und N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[3] W. Aiello, F. Chung und L. Lu. A random graph model for massive graphs. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, Seiten 171–180. ACM, 2000.

[4] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond und U. Stege. A refined search tree technique for dominating set on planar graphs. *Journal of Computer and System Sciences*, 71(4):385 – 405, 2005.

[5] R. Albert und A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[6] C. Banderier, R. Beier und K. Mehlhorn. Smoothed analysis of three combinatorial problems. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Seiten 198–207, 2003.

[7] C. Banderier, H.-K. Hwang, V. Ravelomanana und V. Zacharovas. Analysis of an exhaustive search algorithm in random graphs and the $n^{c \log n}$-asymptotics. *SIAM J. Discrete Math*, 28(1):342–371, 2014.

[8] M. Behrisch und A. Taraz. Efficiently covering complex networks with cliques of similar vertices. *Theoretical Computer Science*, 355(1):37–47, 2006.

[9] S. Ben-David, B. Chor und O. Goldreich. On the theory of average case complexity. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, Seiten 204–216, New York, NY, USA, 1989. ACM. ISBN 0-89791-307-8.

[10] S. Ben-David, B. Chor, O. Goldreich und M. Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992.

[11] D. Bertsimas, C.-P. Teo und R. Vohra. *On dependent randomized rounding algorithms*, Seiten 330–344. Springer Berlin Heidelberg, 1996.

[12] M. Blanchette, E. Kim und A. Vetta. Clique cover on sparse networks. In *Proceedings of the 14th Meeting on Algorithm Engineering & Experiments (ALENEX)*, Seiten 93–102. SIAM, 2012.

[13] M. Bläser und B. Manthey. Smoothed complexity theory. *ACM Transactions on Computation Theory*, 7(2):6, 2015.

[14] J. Blot, W. F. de la Vega, V. T. Paschos und R. Saad. Average case analysis of greedy algorithms for optimisation problems on set systems. *Theoretical Computer Science*, 147(1):267 – 298, 1995.

[15] N. Blum. *Theoretische Informatik*. Oldenbourg, 1998.

[16] H. L. Bodlaender, R. G. Downey, M. R. Fellows und D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8): 423–434, 2009.

[17] B. Bollobás. *Random Graphs*, Seiten 215–252. Springer New York, 1998.

[18] B. Bollobás, P. Erdős, J. Spencer und D. B. West. Clique coverings of the edges of a random graph. *Combinatorica*, 13(1):1–5, 1993.

[19] R. V. Book, K.-I. Ko, S. R. Mahaney und K. McAloon. *Studies in Complexity Theory*. Pitman London, 1986.

[20] C. Boucher und K. Wilkie. Why large closest string instances are easy to solve in practice. In *String Processing and Information Retrieval*, volume 6393, Seiten 106–117. Springer Berlin Heidelberg, 2010.

[21] K. Bringmann, T. Friedrich und A. Krohmer. De-anonymization of heterogeneous random graphs in quasilinear time. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, Seiten 197–208, 2014.

[22] L. Bulteau, F. Hüffner, C. Komusiewicz und R. Niedermeier. Multivariate algorithmics for np-hard string problems. *Bulletin of the EATCS*, 114, 2014.

[23] L. Cai und D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.

[24] M.-S. Chang und H. Müller. On the tree-degree of graphs. In *Graph-theoretic concepts in computer science (Boltenhagen, 2001)*, Seiten 44–54, 2001.

[25] J. Chen, X. Huang, I. A. Kanj und G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72 (8):1346–1367, 2006.

[26] J. Chen, I. A. Kanj und G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

[27] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[28] A. Coja-Oghlan, U. Feige, A. M. Frieze, M. Krivelevich und D. Vilenchik. On smoothed k-CNF formulas and the walksat algorithm. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 451–460, 2009.

[29] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, Seiten 151–158, 1971.

[30] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[31] M. Cygan, M. Pilipczuk und M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 1044–1053. SIAM, 2013.

[32] M. Cygan, S. Kratsch, M. Pilipczuk, M. Pilipczuk und M. Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Transactions on Computation Theory*, 6(2):6, 2014.

[33] I. Dinur und S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[34] R. G. Downey und M. R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[35] R. G. Downey und M. R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.

[36] R. G. Downey und M. R. Fellows. *Parameterized Complexity*. Springer-Verlag New York, 1999.

[37] R. G. Downey und M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag London, 2013.

[38] D. P. Dubhashi und A. Panconesi. *Concentration Of Measure For The Analysis of Randomized Algorithms*. Cambridge University Press, 2012.

[39] P. Erdős und A. Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Science*, Seiten 17–61, 1960.

[40] K. Erk und L. Priese. *Theoretische Informatik*. Springer-Verlag Berlin Heidelberg, 2002.

[41] H. Fernau. Parameterized algorithmics for d-hitting set. *International Journal of Computer Mathematics*, 87(14):3157–3174, 2010.

[42] J. Flum und M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag Berlin Heidelberg, 2006.

[43] N. Fountoulakis, T. Friedrich und D. Hermelin. On the average-case complexity of parameterized clique. *Theoretical Computer Science*, 576:18–29, 2015.

[44] T. Friedrich und C. Hercher. On the kernel size of clique cover reductions for random intersection graphs. *Journal of Discrete Algorithms*, 34:128–136, 2015.

[45] T. Friedrich und A. Krohmer. Parameterized clique on scale-free networks. In *Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC)*, Seiten 659–668, 2012.

[46] T. Friedrich und A. Krohmer. Parameterized clique on inhomogeneous random graphs. *Discrete Applied Mathematics*, 184:130–138, 2015.

[47] T. Friedrich, T. Sauerwald und D. Vilenchik. Smoothed analysis of balancing networks. *Random Structures & Algorithms*, 39(1):115–138, 2011.

[48] A. Frieze und B. Reed. Covering the edges of a random graph by cliques. *Combinatorica*, 15(4):489–497, 1995.

[49] W. Gao, T. Friedrich und F. Neumann. Fixed-parameter single objective search heuristics for minimum vertex cover. In *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*, Seiten 740–750, 2016.

[50] M. R. Garey und D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[51] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[52] T. F. Gonzalez. Approximation algorithms and metaheuristics. In *Computing Handbook, Third Edition: Computer Science and Software Engineering*, Seiten 12: 1–26. 2014.

[53] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H. Piepho und R. Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(2):725–736, 2007.

[54] J. Gramm, J. Guo, F. Hüffner und R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13: 2.2, 2009.

[55] T. Grossman und A. Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81 – 92, 1997.

[56] J.-L. Guillaume und M. Latapy. Bipartite structure of all complex networks. *Information Processing Letters*, 90(5):215–221, 2004.

[57] Y. Gurevich. Average case completeness. *J. Comput. Syst. Sci.*, 42(3):346–398, 1991.

[58] A. Gyárfás. A simple lower bound on edge coverings by cliques. *Discrete Mathematics*, 85(1):103–104, 1990.

[59] D. Hermelin, R. Rizzi und S. Vialette. Algorithmic aspects of the intersection and overlap numbers of a graph. In *Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC)*, Seiten 465–474, 2012.

[60] D. Hoover. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 11:187–208, 1992.

[61] J. Hromkovič. *Theoretische Informatik – Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kryptographie. Eine Einführung.* Teubner, 2004.

[62] J. Hromkovič. *Algorithmic Adventures – From Knowledge to Magic.* Springer-Verlag Berlin Heidelberg, 2009.

[63] R. Impagliazzo und R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[64] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, Seiten 38–49, New York, NY, USA, 1973. ACM.

[65] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller und J. W. Thatcher, editors, *Complexity of Computer Computations*, Seiten 85–103. Plenum Press, 1972.

[66] V. Klee und G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, Seite 159–175. Academic Press, New York, 1972.

[67] D. E. Knuth. *The Art Of Computer Programming Volume 3 – Sorting and Searching.* Addison-Wesley, 1973.

[68] L. T. Kou, L. J. Stockmeyer und C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM*, 21(2):135–139, 1978.

[69] M. Krivelevich, D. Reichman und W. Samotij. Smoothed analysis on connected graphs. *SIAM J. Discrete Math.*, 29(3):1654–1669, 2015.

[70] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265 –266, 1973.

[71] L. A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, Feb. 1986.

[72] N. Livne. All natural np-complete problems have average-case complete versions. *computational complexity*, 19(4):477–499, 2010.

[73] L. Lu und X. Peng. Spectra of edge-independent random graphs. *Electr. J. Comb.*, 20(4):P27, 2013.

[74] C. Lund und M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[75] S. Ma, W. D. Wallis und J. Wu. Clique covering of chordal graphs. *Utilitas Mathematica*, 36:151–152, 1989.

[76] K. Michal, E. R. Scheinerman und K. B. Singer-Cohen. On random intersection graphs: The subgraph problem. *Combinatorics, Probability and Computing*, 8:131–159, 1999.

[77] R. Motwani und P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[78] N. Mousavi. How tight is chernoff bound? `https://ece.uwaterloo.ca/~nmousavi/Papers/Chernoff-Tightness.pdf`, 2010.

[79] M. Müller. *Parameterized Randomization*. PhD thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 2008.

[80] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2008.

[81] R. Niedermeier und P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89 – 102, 2003. ISSN 1570-8667. Combinatorial Algorithms.

[82] J. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

[83] C. H. Papadimitriou und M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425 – 440, 1991.

[84] F. Papadopoulos, M. Kitsak, M. Serrano, M. Boguñá und D. Krioukov. Popularity versus Similarity in Growing Networks. *Nature*, 489:537–540, Sep 2012.

[85] D. Peleg, G. Schechtman und A. Wool. Randomized approximation of bounded multicovering problems. *Algorithmica*, 18(1):44–66, 1997.

[86] H.-P. Piepho. An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13(2):456–466, 2004.

[87] M. Pilipczuk. *Tournaments and Optimality: New Results in Parameterized Complexity*. PhD thesis, University of Bergen, Norway, 2013.

[88] P. Raghavan und C. D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[89] S. Rajagopalan, M. Vachharajani und S. Malik. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *Proceedings of the International Conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, Seiten 157–164, 2000.

[90] J. M. Robson. Finding a maximum independent set in time $\mathcal{O}(2^{n/4})$. Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001.

[91] S. Sakai, M. Togasaki und K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2–3):313 – 322, 2003.

[92] U. Schöning. *Theoretische Informatik – kurz gefasst*. Spektrum Akademischer Verlag, 2009.

[93] E. V. Slud. Distribution inequalities for the binomial law. *Ann. Probab.*, 5(3): 404–412, 1977.

[94] D. A. Spielman und S.-H. Teng. Smoothed analysis (motivation and discrete models). In *8th International Workshop on Algorithms and Data Structures (WADS)*, Seiten 256–270, 2003.

[95] D. A. Spielman und S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, May 2004.

[96] R. van Bevern, R. Bredereck, M. Chopin, S. Hartung, F. Hüffner, A. Nichterlein und O. Suchý. *Parameterized Complexity of DAG Partitioning*, Seiten 49–60. Springer Berlin Heidelberg, 2013.

[97] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Linköpings universitet, Sweden, 2007.

[98] J. Wang. Average-case computational complexity theory. In *Complexity Theory Retrospective II*, Seiten 295–328. Springer New York, 1997.

[99] G. Wechsung. *Vorlesungen zur Komplexitätstheorie*. Teubner, 2000.

[100] H. S. Wilf. Backtrack: An $\mathcal{O}(1)$ expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18(3):119 – 121, 1984.

[101] J. Zhao, O. Yagan und V. Gligor. Random intersection graphs and their applications in security, wireless communication, and social networks. In *Proceedings of the Information Theory and Applications Workshop (ITA)*, 2015.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass mir die Promotionsordnung der Fakultät für Mathematik und Informatik der Friedrich- Schiller-Universität Jena bekannt ist,

ich die Dissertation selbst angefertigt habe und alle von mir benutzten Hilfsmittel, persönlichen Mitteilungen und Quellen in meiner Arbeit angegeben sind,

mich die folgenden Personen bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts unterstützt haben: Prof. Dr. Tobias Friedrich und Prof. Dr. Martin Mundhenk,

die Hilfe eines Promotionsberaters nicht in Anspruch genommen wurde und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,

dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe und

dass ich die gleiche, eine in wesentlichen Teilen ähnliche oder eine andere Abhandlung nicht bei einer anderen Hochschule als Dissertation eingereicht habe.

Jena, 29. 05. 2017 Christian Hercher